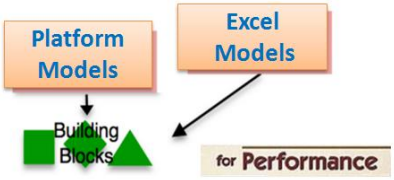
Quick Start Guide – TurboVisualStudio

TurboActuary Ecosystem 3.0



TurboActuary tA#

©2013 by TurboActuary LLC.
All Rights Reserved

CHAPTER 1: INSTALLATION 1

1.1 INSTALLATION OF VISUAL STUDIO 2010/2012/2013 1

1.2 UNBLOCK DLL 1

1.3 TURBOVISUALIZER AND TURBOINTELLISENSE INSTALLATION 2

CHAPTER 2: CREATE YOUR OWN EXCEL FUNCTIONS 4

2.1 WRITING ONCE AND USE EVERYWHERE 4

2.2 VIEW C# CODE FROM EXCEL 9

CHAPTER 3: PLATFORM MODELS 12

3.1 HIERARCHICAL ARCHITECTURE 12

3.2 BUILDING MODELS INCREMENTALLY 13

3.3 TRACE MAIN FUNCTION EXECUTION 14

3.4 HIGH PERFORMANCE IN MULTI-CPU/CORE COMPUTERS 18

3.5 DEPENDENCE TRACING 19

3.6 DEPENDENCE DEBUGGING 21

3.7 TURBO NAVIGATION 22

3.8 PRINT THE WHOLE CLASS 24

CHAPTER 4: TURBO INTELLIGENT 25

4.1 HIGHLIGHT 25

4.2 AUTO-GENERATE PARAMETERS 27

4.3 AUTO-GENERATE CALCULATION DRIVER 27

4.4 AUTO-GENERATE EXCEL MODEL 31

CHAPTER 5: CUSTOMIZE THE MODEL FOR YOUR BUSINESS 34

5.1 ADD A NEW ASSUMPTION TABLE INTO INPUT FILE 34

5.2 Read in the Assumption table as a Pivot Table into PlanSpec 35

5.3 Add Persistence Bonus Business Logic 35

5.4 Update Beginning Cash Flow Function 36

5.5 Generate the Calculation Driver 37

5.6 Recompiling and Running in VS 38

5.7 Generate Excel Model 38

5.8 View Excel Model Formula C# Code 42

5.9 Output Results 43

APPENDIX 1: CODING REUSE/INHERITANCE 45

APPENDIX 2: HOW TO USE VISUAL STUDIO 46

2.1 Quick Launch in VS 2012 46

2.2 IntelliSense 46

2.3 Coding Navigation 48

2.4 Debugging 53

2.5 Docking Window 57

2.6 Diff Tool 58

2.7 Useful Shortcut 58

APPENDIX 3: ABOUT C# 60

APPENDIX 4: VISUAL STUDIO TOOLS 61

4.1 Blue Theme (Optional) 61

4.2 Indent Guides (Optional) 61

Chapter 1: Installation

Please read Quick Study Guide – TurboExcel first.

If you run into problems, please read **Appendix 3 of Quick Study Guide – TurboExcel**.

1.1 Installation of Visual Studio 2010/2012/2013

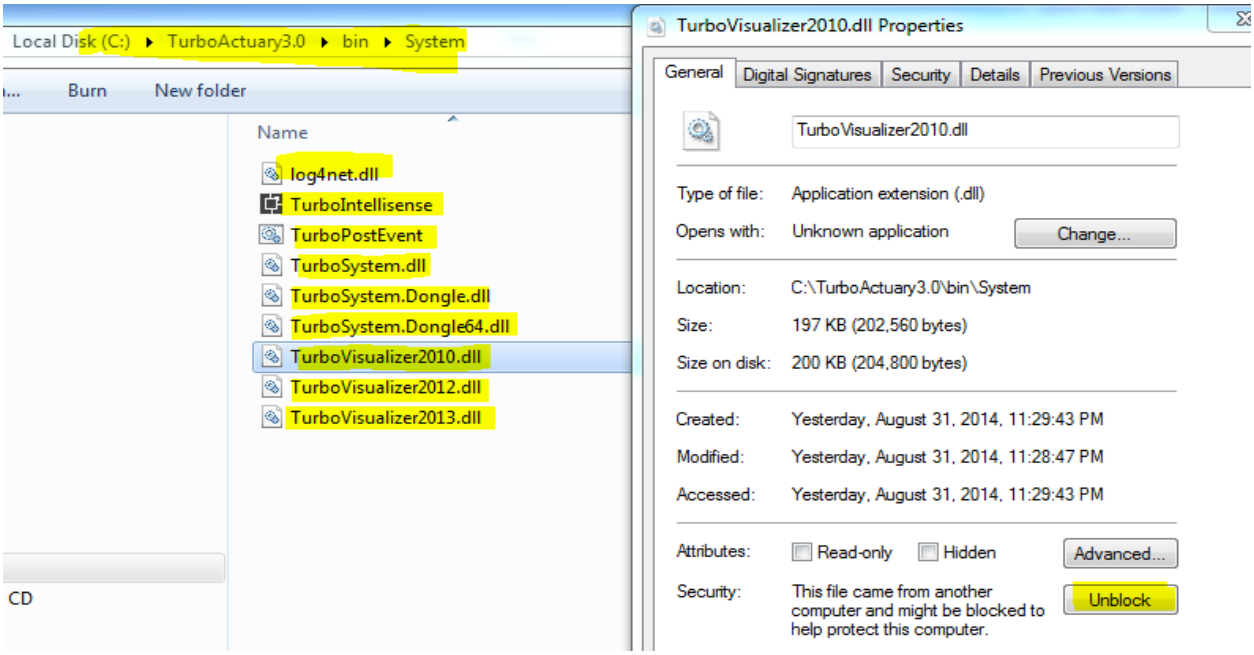
You can install VS 2010/2012/2013 Professional or above.

They are 90 days free at http://www.microsoft.com/visualstudio/eng/downloads.

Your company can purchase VS 2012 Professional much cheaper than its retail price (US$ 499 without MSDN subscription) by using volume licensing sales channel.

1.2 Unblock DLL

Unblock these files in C:\TurboActuary3.0\bin\System after righ click and choose propertis:



1.3 TurboVisualizer and TurboIntellisense Installation

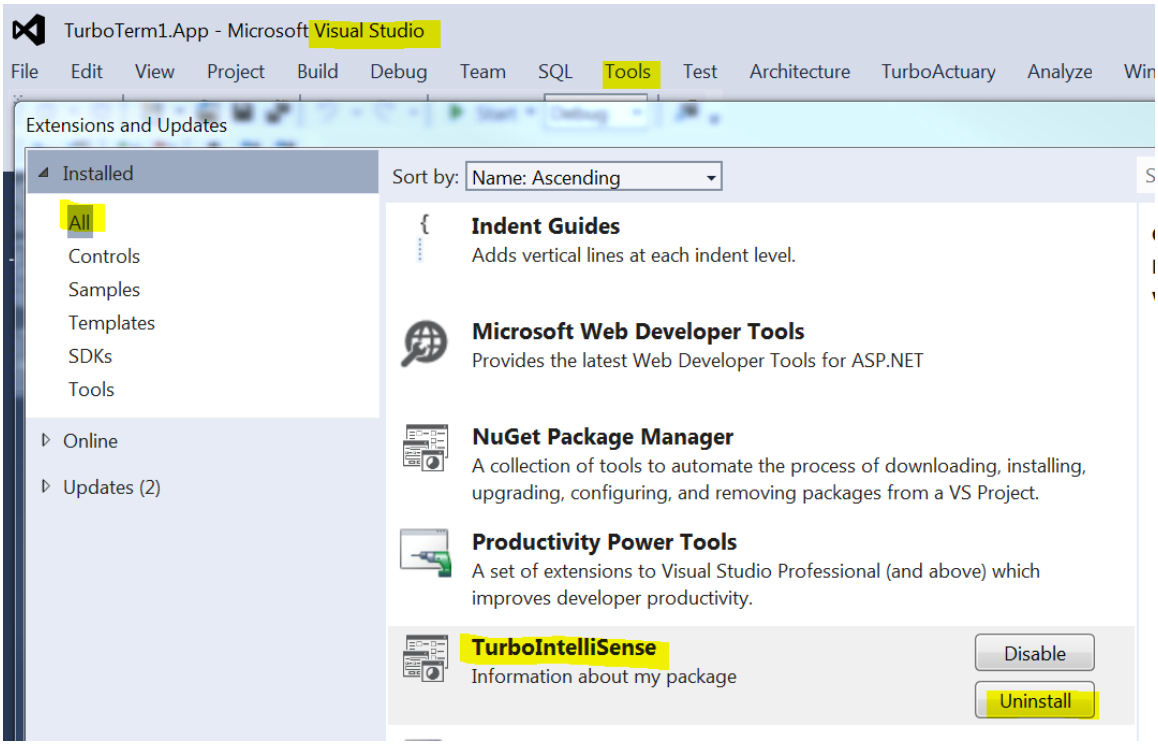
a. For VS 2010, copy C:\TurboActuary3.0\bin\System\TurboVisualizer2010.dll to My Documents\Visual Studio 2010\Visualizers.

b. For VS 2012/2013, do similar as a).

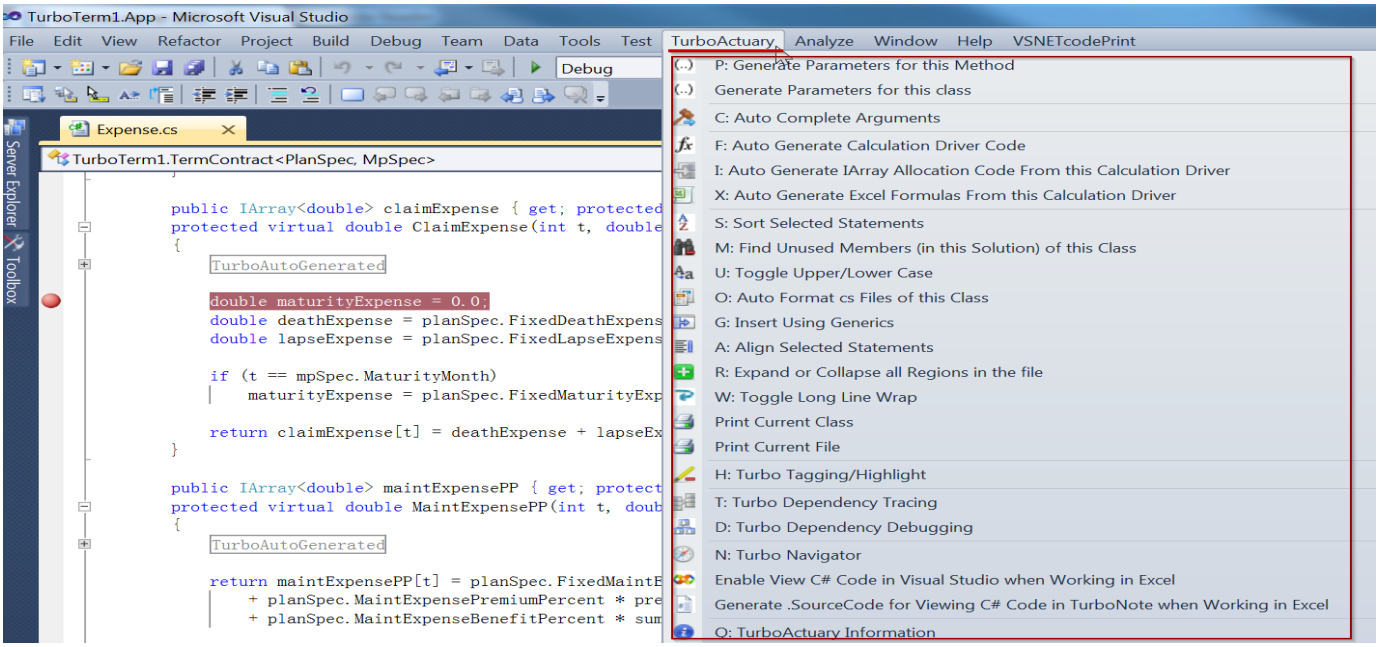
c. Double-Click to open and install TurboIntellisense.vsix in C:\TurboActuary3.0\bin\System (you might need to select Microsoft Visual Studio Version Selector to open it).

Trouble-Shooting: If visualizers subfold does not exist, then create it and do (a) and (b).

Trouble-Shooting: if you have installed TurboIntellisense.vsix before, you need to uninstall it first, and then reinstall. To uninstall it, choose "Extensions and Updates" submenu in in tools menu, and then choose TurboIntelliSense and uninstall.

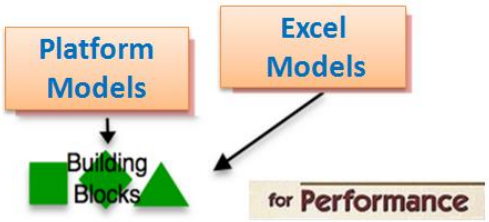

TurboTerm1.App - Microsoft Visual Studio
File Edit View Project Build Debug Team SQL Tools Test Architecture TurboActuary Analyze Win
Extensions and Updates
Installed
All
Controls
Samples
Templates
SDKs
Tools
Online
Updates (2)
Sort by: Name: Ascending
Indent Guides
Adds vertical lines at each indent level.
Microsoft Web Developer Tools
Provides the latest Web Developer Tools for ASP.NET
NuGet Package Manager
A collection of tools to automate the process of downloading, installing, upgrading, configuring, and removing packages from a VS Project.
Productivity Power Tools
A set of extensions to Visual Studio Professional (and above) which improves developer productivity.
TurboIntelliSense
Information about my package
Disable
Uninstall

If you open Visual Studio, and you will see TurboActury main menu and submenus:


TurboTerm1.App - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Test TurboActuary Analyze Window Help VSNETcodePrint
Debug
Expense.cs
TurboTerm1.TermContract<PlanSpec, MpSpec>
public IArray<double> claimExpense { get; protected
protected virtual double ClaimExpense(int t, double
{
TurboAutoGenerated
double maturityExpense = 0.0;
double deathExpense = planSpec.FixedDeathExpens
double lapseExpense = planSpec.FixedLapseExpens
if (t == mpSpec.MaturityMonth)
maturityExpense = planSpec.FixedMaturityExp
return claimExpense[t] = deathExpense + lapseEx
}
public IArray<double> maintExpensePP { get; protect
protected virtual double MaintExpensePP(int t, doub
{
TurboAutoGenerated
return maintExpensePP[t] = planSpec.FixedMaintE
+ planSpec.MaintExpensePremiumPercent * pre
+ planSpec.MaintExpenseBenefitPercent * sum
Server Explorer
Toolbox
P: Generate Parameters for this Method
Generate Parameters for this class
C: Auto Complete Arguments
F: Auto Generate Calculation Driver Code
I: Auto Generate IArray Allocation Code From this Calculation Driver
X: Auto Generate Excel Formulas From this Calculation Driver
S: Sort Selected Statements
M: Find Unused Members (in this Solution) of this Class
U: Toggle Upper/Lower Case
O: Auto Format cs Files of this Class
G: Insert Using Generics
A: Align Selected Statements
R: Expand or Collapse all Regions in the file
W: Toggle Long Line Wrap
Print Current Class
Print Current File
H: Turbo Tagging/Highlight
T: Turbo Dependency Tracing
D: Turbo Dependency Debugging
N: Turbo Navigator
Enable View C# Code in Visual Studio when Working in Excel
Generate .SourceCode for Viewing C# Code in TurboNote when Working in Excel
O: TurboActuary Information

Chapter 2: Create your own Excel functions

Before we show you our platform models, we would like to show you first how easily you can create your own c# functions. Please note that any function you created can be used by both platform models and Excel models without any extra work.



For those people who are not familiar with C#: C# is the most popular language in windows platform, and please refer to Appendix 3 for more detail.

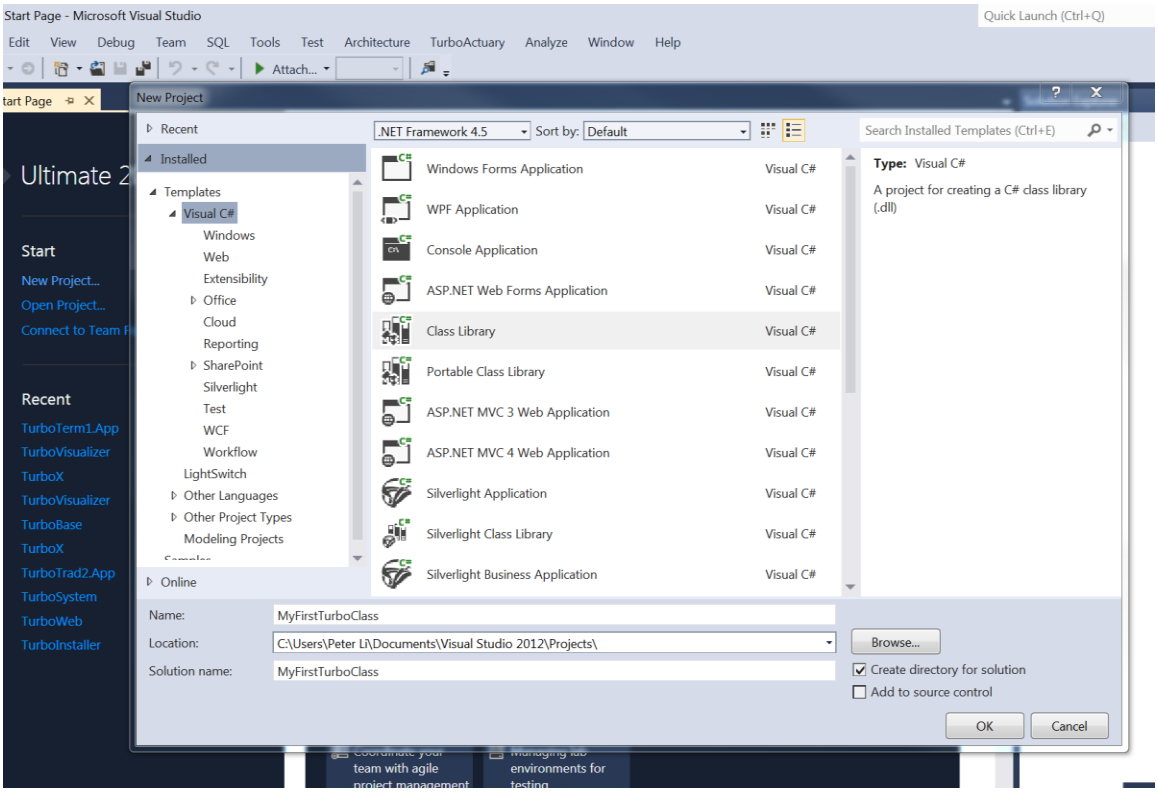
The exercise in this chapter is similar as

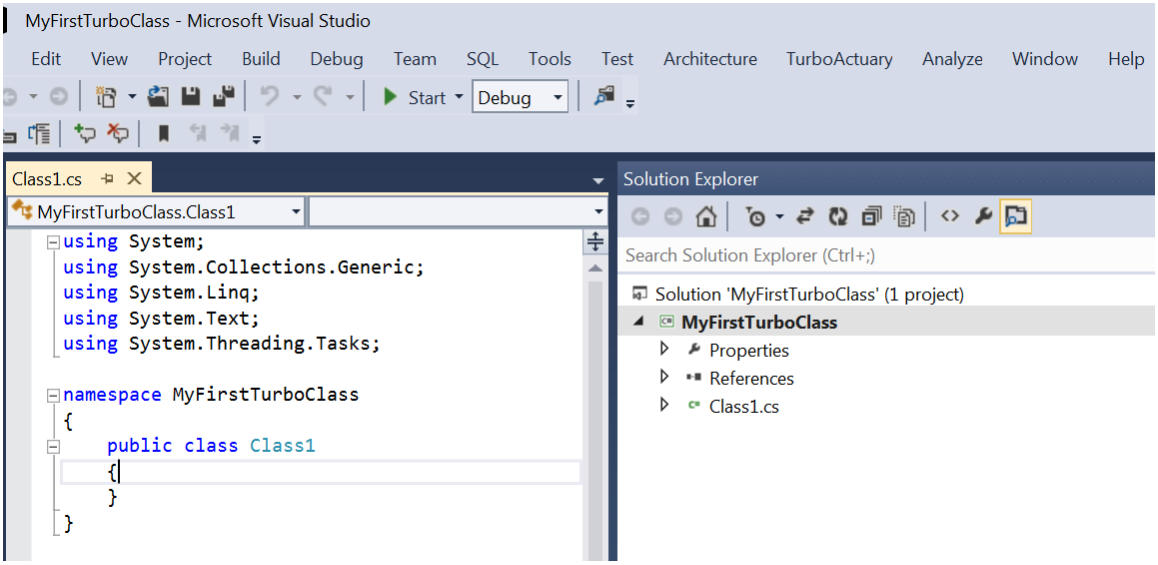
a) the solution under C:\TurboActuary3.0\Application\CSharp\Exercise\Exercise1

b) Exerice.xlsb under C:\TurboActuary3.0\Application\Excel – Math.

2.1 Writing Once and Use Everywhere

Now open Visual Studio, choose New Project, then C#, then Class Library, and type "MyFirstTurboClass", and click OK. It will look like:





Please write several functions inside the class. It should look like:

Class1.cs*

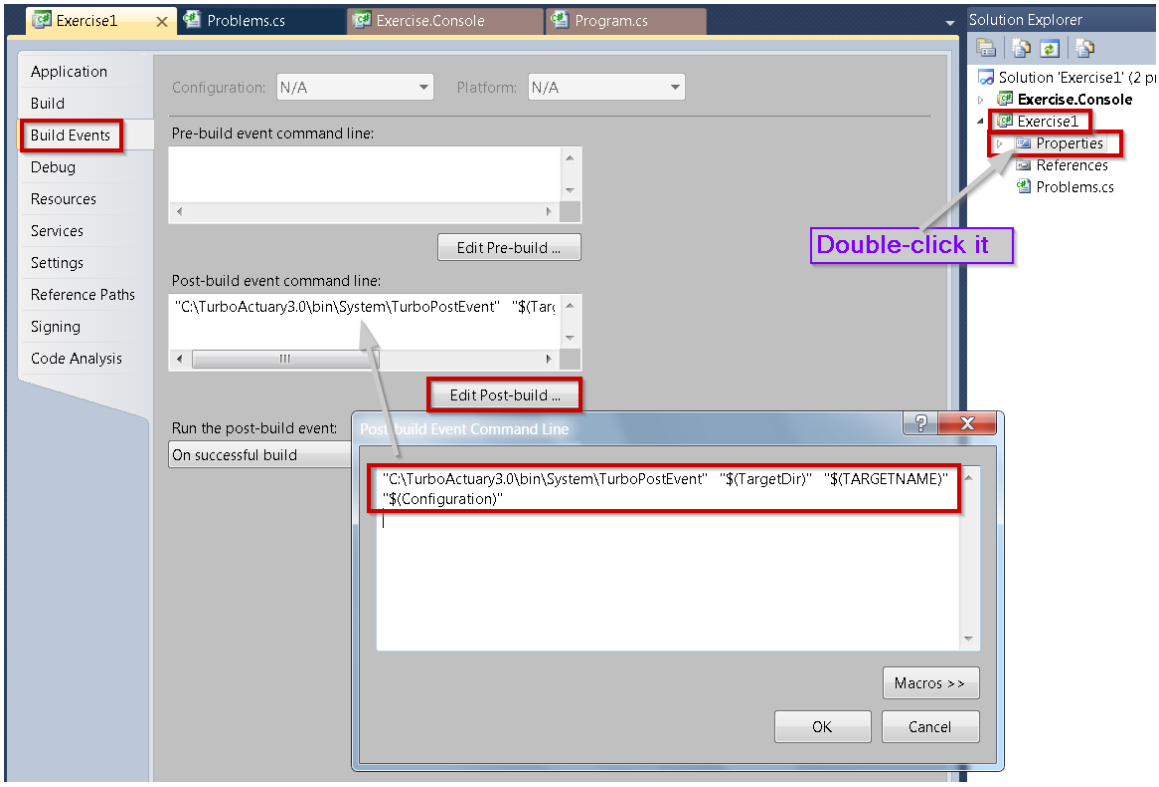
MyFirstTurboClass.Class1 — ThisYearValue(double interestRate)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyFirstTurboClass
{
    public class Class1
    {
        private double lastYearValue;

        public Class1(double lastYearValue)
        {
            this.lastYearValue = lastYearValue;
        }

        public static string Hello()
        {
            return "Hello Actuary!";
        }

        public double ThisYearValue(double interestRate)
        {
            return lastYearValue * (1.0 + interestRate);
        }
    }
}
```

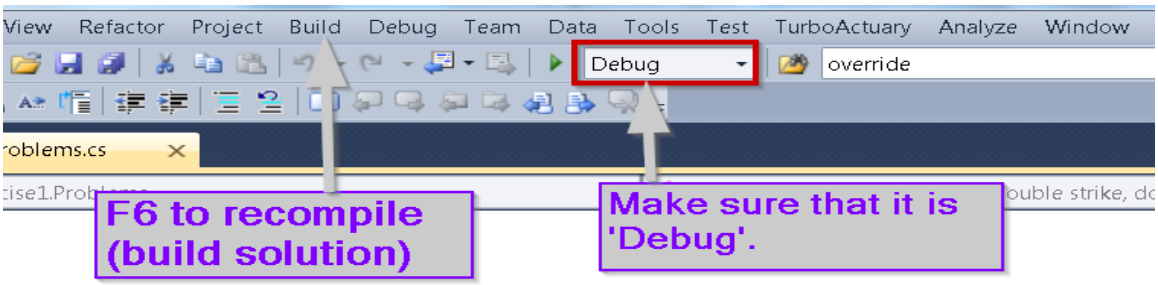
In the Solution Explore, double-click the Properties, then go to “Build Event”, click “Edit Post-build”, and put the one copy command inside the input box:

"C:\TurboActuary3.0\bin\System\TurboPostEvent" "$(TargetDir)" "$(TARGETNAME)" "$(Configuration)"

It should look like:



Click OK. Then Save the solution. Then press F6 (or click Build Solution submenu under Build menu) to compile and build the solution **in debug mode**. Correct any typing error if build not succeed.

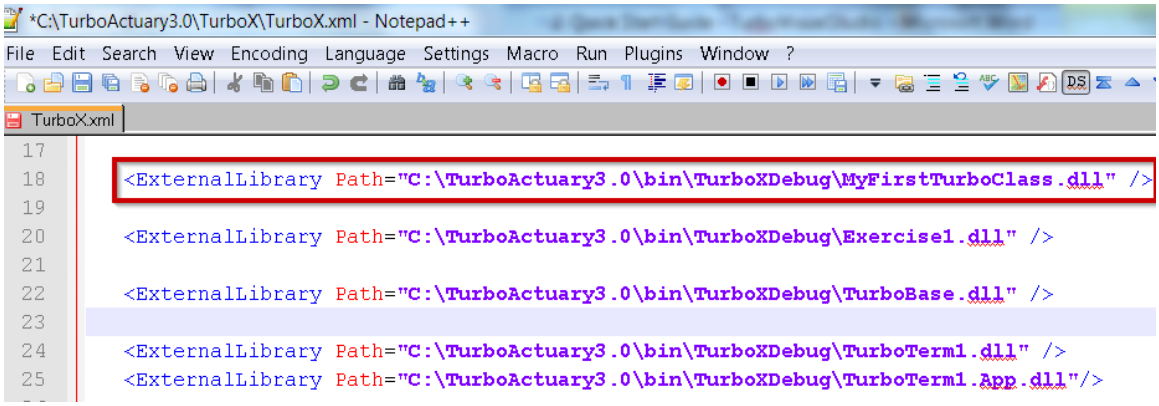

View Refactor Project Build Debug Team Data Tools Test TurboActuary Analyze Window
Debug
override
F6 to recompile (build solution)
Make sure that it is 'Debug'.

```
public static double  CallPrice(double spot, double strike, double volatility,
double r)
{
    double a = Math.Log(spot / strike);
    double b_call = (r  + 0.5 * Math.Pow(volatility, 2)) * t;
    double b_put = (r  - 0.5 * Math.Pow(volatility, 2)) * t;
    double c = volatility * Math.Sqrt(t);
    double d1 = (a + b_call) / c;
    double d2 = (a + b_put) / c;
    CumulativeNormalDistribution cumulativeNormalDistribution = new
    CumulativeNormalDistribution();
    double Price = spot * cumulativeNormalDistribution.Value(d1)
        - strike * Math.Exp(-r * t) * cumulativeNormalDistribution.Value(d2);
    return Price;
}
```

Trouble-Shooting: if VS complains that the TurboPostEvent fails, close all Excel books. Then restart Visual Studio, and retry.

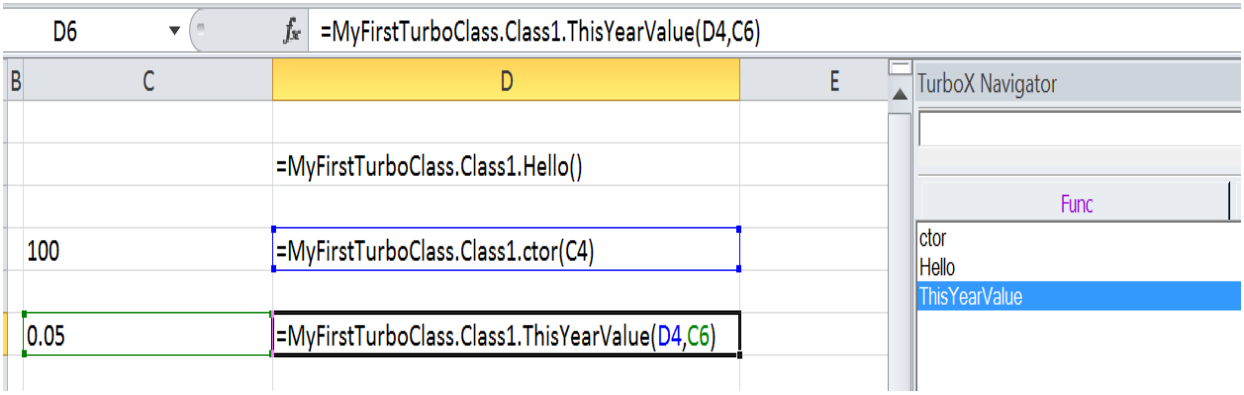
Now you have finished our first library with 3 functions inside 1 class. You are going to use them in Excel.

Use Notepad++ (or Notepad) to open C:\TurboActuary3.0\TurboX\TurboX.xml. Insert <ExternalLibrary Path="C:\TurboActuary3.0\bin\TurboXDebug\MyFirstTurboClass.dll" /> into TurboX.xml, and it should look like:

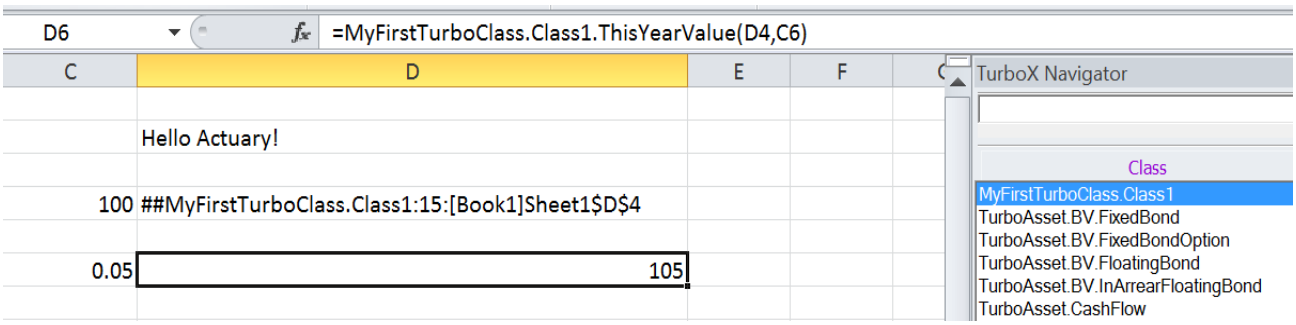


Save and close it.

Open a blank Excel workbook, and type the three formulas (or use TurboX Navigator) as following:



The three formulas will have results like:



Now you should have some sense how easily you can write your high-performance building blocks which can be shared by both platform model and Excel model.

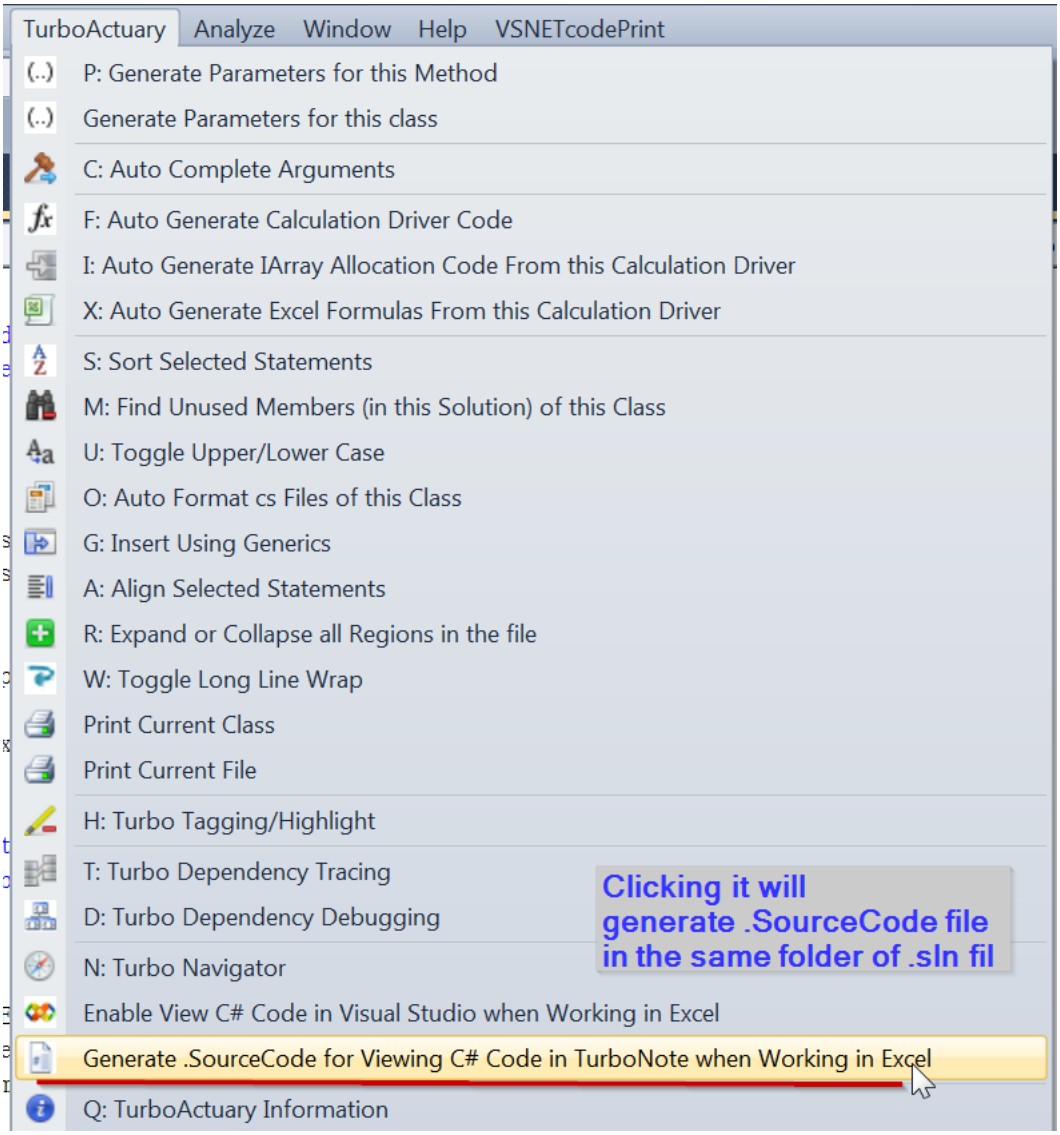
2.2 View C# Code From Excel

You can use Visual Studio to **view and debug** the C# codes when you are working in Excel as we have discussed in the chapter 4 of Quick Start Guide – TurboExcel.

Alternatively you can use TurboNote or Notepad++ to **view** C# codes when you are working in Excel as we discussed in the section 2.3. However, this requires you or somebody else has done the followings:

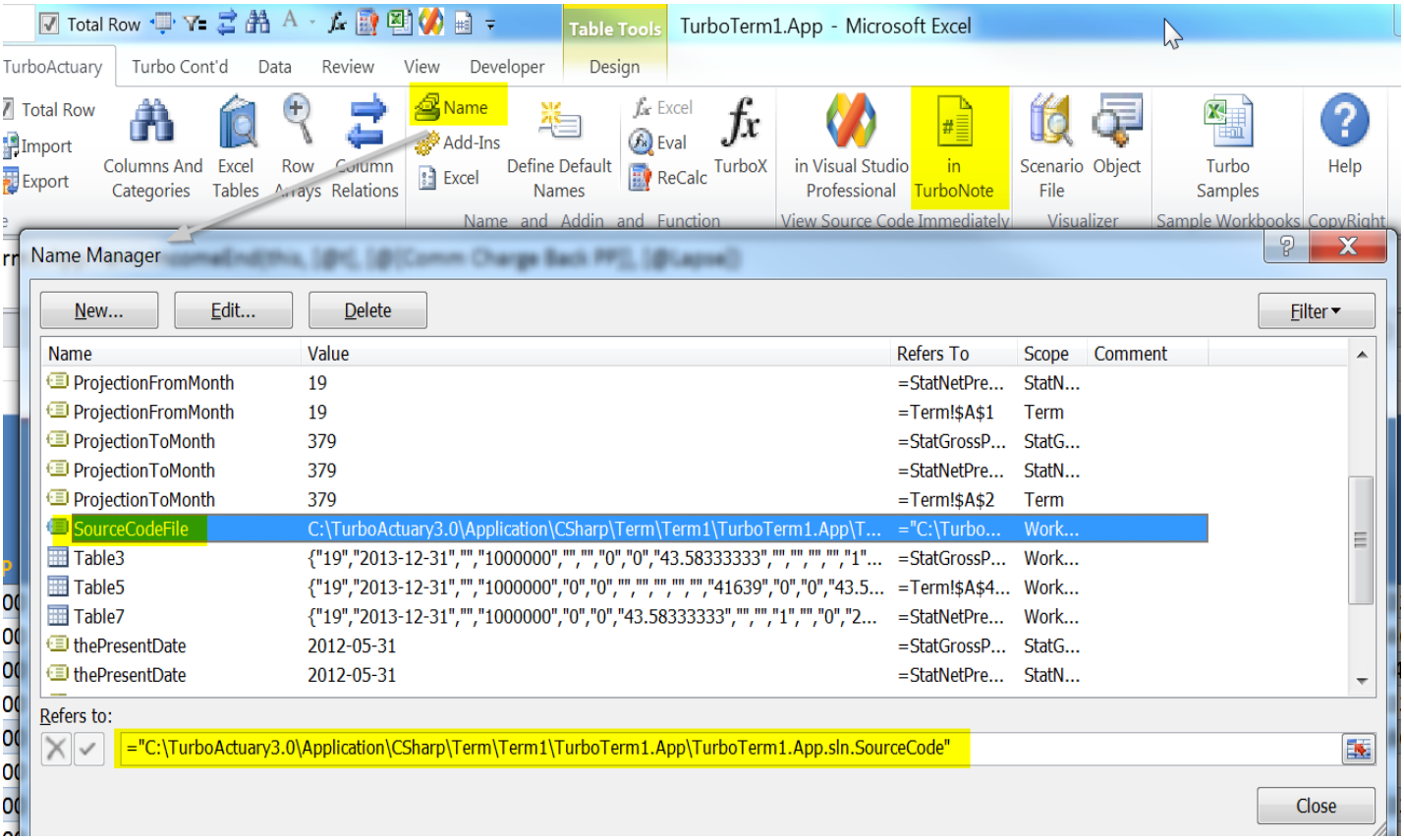
1) Generate .SourceCode file in Visual Studio.

   Remember the path and name of the generated file, that will be used in step 2.


TurboActuary
Analyze
Window
Help
VSNETcodePrint
P: Generate Parameters for this Method
Generate Parameters for this class
C: Auto Complete Arguments
F: Auto Generate Calculation Driver Code
I: Auto Generate IArray Allocation Code From this Calculation Driver
X: Auto Generate Excel Formulas From this Calculation Driver
S: Sort Selected Statements
M: Find Unused Members (in this Solution) of this Class
U: Toggle Upper/Lower Case
O: Auto Format cs Files of this Class
G: Insert Using Generics
A: Align Selected Statements
R: Expand or Collapse all Regions in the file
W: Toggle Long Line Wrap
Print Current Class
Print Current File
H: Turbo Tagging/Highlight
T: Turbo Dependency Tracing
D: Turbo Dependency Debugging
N: Turbo Navigator
Enable View C# Code in Visual Studio when Working in Excel
Generate .SourceCode for Viewing C# Code in TurboNote when Working in Excel
Q: TurboActuary Information
Clicking it will
generate .SourceCode file
in the same folder of .sln fil

2) Define SourceCodeFile name in the name manager of your spreadsheet. Its value will be the full file name (with path) of the generated file in step 1. Then you can view all source codes in the solution when you are working in Excel.

   For example, in our TurboTerm1.App.xlsb, we have


Total Row
Table Tools
TurboTerm1.App - Microsoft Excel
TurboActuary
Turbo Cont'd
Data
Review
View
Developer
Design
Total Row
Import
Export
Columns And Categories
Excel Tables
Row Arrays
Column Relations
Name
Add-Ins
Excel
Define Default Names
Excel
Eval
ReCalc
TurboX
in Visual Studio Professional
in TurboNote
Scenario File
Object
Turbo Samples
Help
Name and Addin and Function
View Source Code Immediately
Visualizer
Sample Workbooks
CopyRight
Name Manager
New...
Edit...
Delete
Filter
Name
Value
Refers To
Scope
Comment
ProjectionFromMonth 19 =StatNetPre... StatN...
ProjectionFromMonth 19 =Term!$A$1 Term
ProjectionToMonth 379 =StatGrossP... StatG...
ProjectionToMonth 379 =StatNetPre... StatN...
ProjectionToMonth 379 =Term!$A$2 Term
SourceCodeFile C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App\T... ="C:\Turbo... Work...
Table3 {"19","2013-12-31","","1000000","","","0","0","43.58333333","","","","","1"... =StatGrossP... Work...
Table5 {"19","2013-12-31","","1000000","0","0","","","","","","41639","0","0","43.5... =Term!$A$4... Work...
Table7 {"19","2013-12-31","","1000000","0","0","43.58333333","","","1","","0","2... =StatNetPre... Work...
thePresentDate 2012-05-31 =StatGrossP... StatG...
thePresentDate 2012-05-31 =StatNetPre... StatN...
Refers to:
="C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App\TurboTerm1.App.sln.SourceCode"
Close

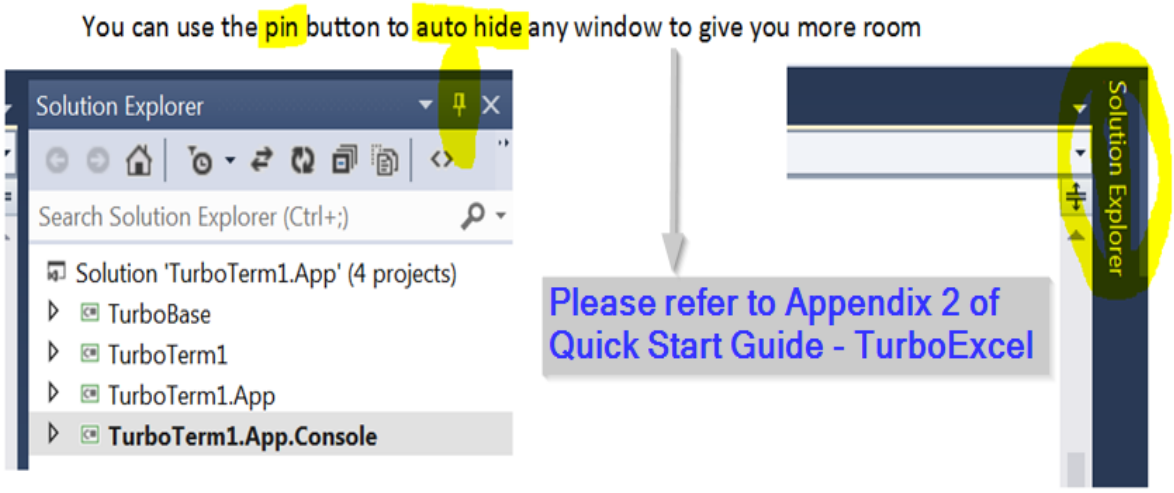
Chapter 3: Platform Models

3.1 Hierarchical Architecture

The library codes are under C:\TurboActuary3.0\Liability while the application codes are under C:\TurboActuary3.0\Application\CSharp.

These applications are built on top of these libraries. The application layer inherits the codes of library layer, so it is "reuse" not "rewrite". A good example of inheritance can be found in the Appendix 1.

Go to C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App, and open TurboTerm1.App.sln. If you do not see the Solution Explorer, you can go to view menu and click Solution Explorer submenu. You will see there are four projects in the solution.


You can use the pin button to auto hide any window to give you more room
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'TurboTerm1.App' (4 projects)
TurboBase
TurboTerm1
TurboTerm1.App
TurboTerm1.App.Console
Please refer to Appendix 2 of Quick Start Guide - TurboExcel
Solution Explorer

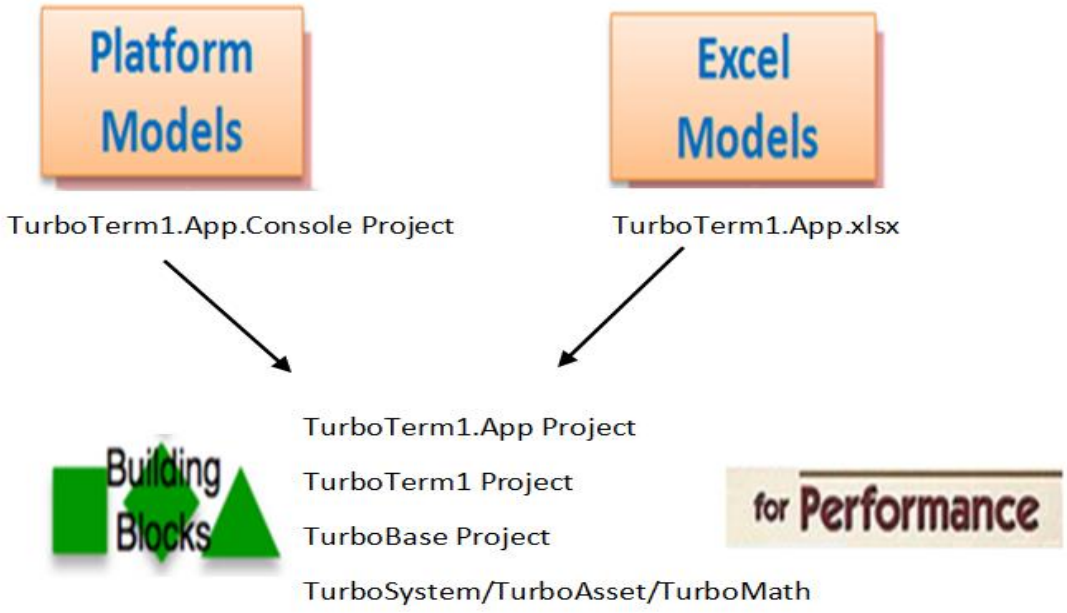
The TurboBase project has classes used by all libraries and applications.

The TurboTerm1 project is one of our libraries. At this layer, we have common business logic and abstract data for Term product.

The TurboTerm1.App project is at library application layer. At this layer, we may decorate or slightly change the business logic for different Term plan, and supply the concrete data into the models.

All of the above three layers are building blocks used by both application models and Excel models.

The TurboTerm1.App.Console project is at user application layer. The console project is parallel to Excel application in our hierarchical architecture.



3.2 Building Models Incrementally

We use Objected-Oriented design, so we can build models incrementally. Please refer to Appendix 1: Coding Reuse/Inheritance.

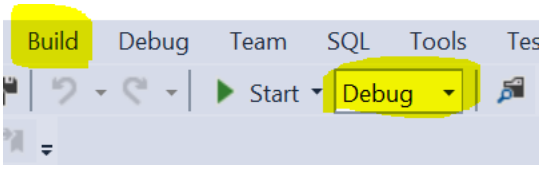
We have provided rich libraries for liability/Asset/ALM. You never have to write your models from scratch. You only need to extend some library classes, and put your customization codes in these new derived classes. So you never mingle the codes of your new classes with the

codes of their base classes. Your derived classes will inherit all functions from their base classes, but you will decorate a few of them with your new code.

That is: we have built all framworks for your mansions, and you will decorate them in your style.

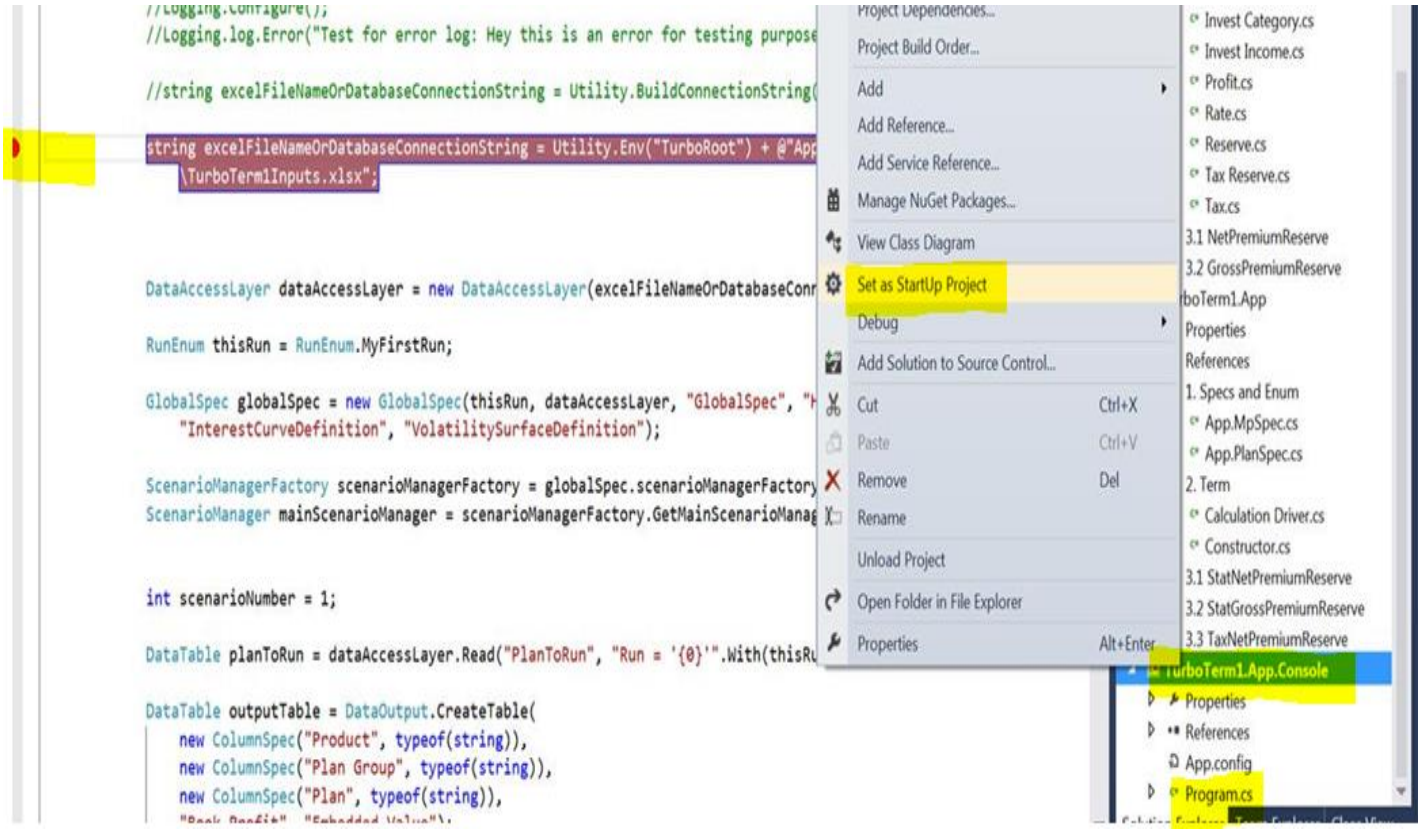
3.3 Trace Main Function Execution

In Debug configuration (see the picture below), hit F6 to rebuild the solution.

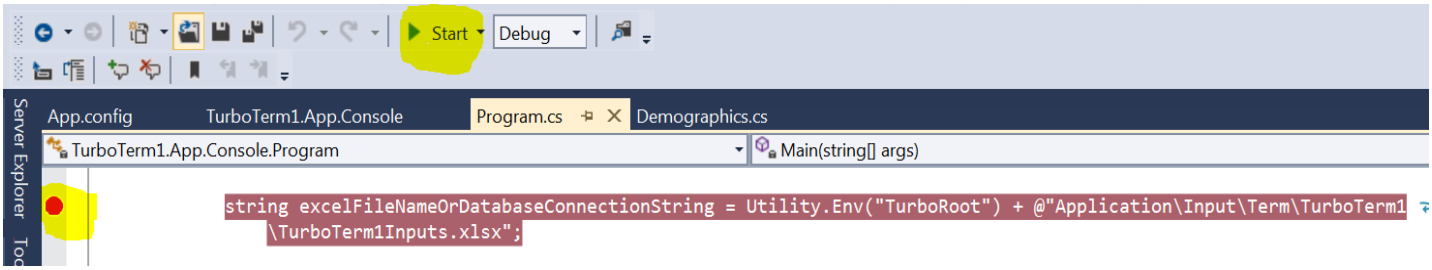


Trouble-Shooting: if VS complains that the copy fails, close all Excel books and retry.

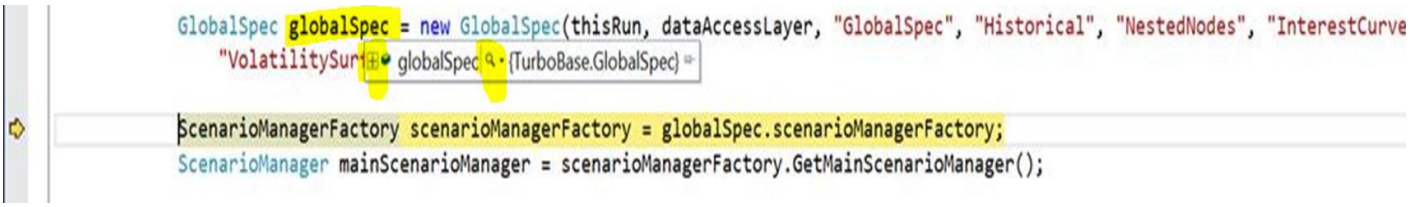
In solution explorer, expand the project TurboTerm1.App.Console, make sure it is the startup project (i.e. it should have bold font). If not, right click the TurboTerm1.App.Console, and set it to be startup project.

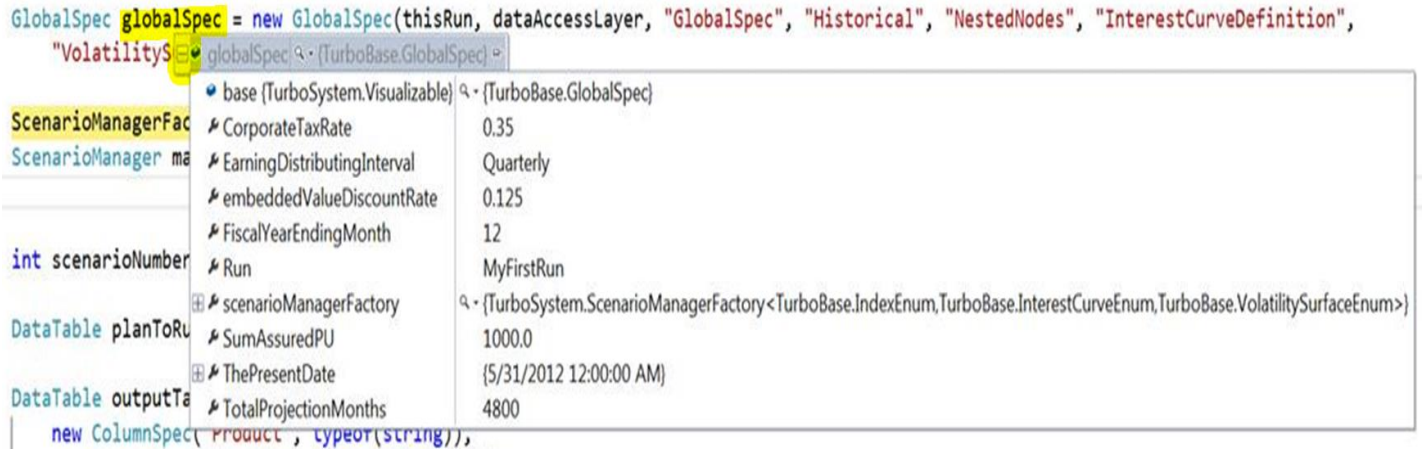


Double-click Program.cs. Set a breakpoint at the beginning of Main function. Then click "Start" (or hit F5) to run it in debug mode.



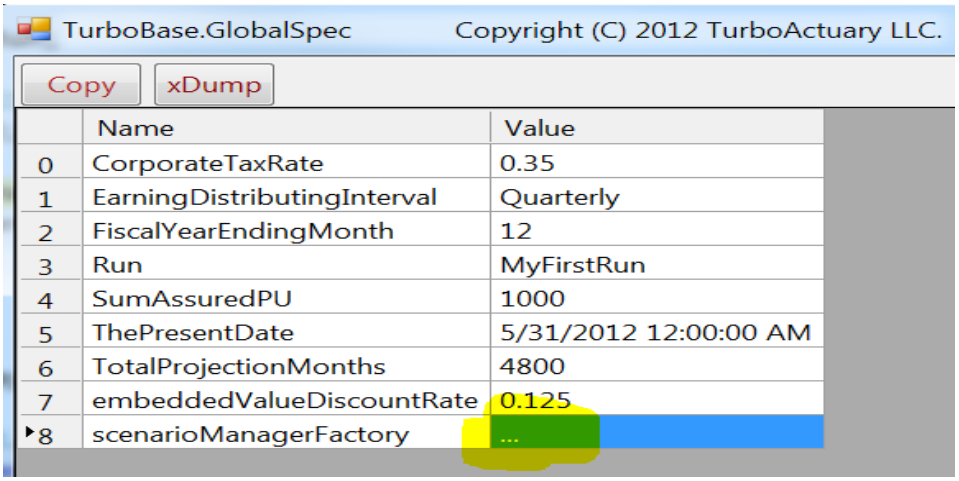
Press F10 to step over, F11 to step into, shift+F11 to step out. To view data, you can hover over the variable name to click the expand icon, or magnifying glass icon as following:


GlobalSpec globalSpec = new GlobalSpec(thisRun, dataAccessLayer, "GlobalSpec", "Historical", "NestedNodes", "InterestCurve
"VolatilitySur
globalSpec {TurboBase.GlobalSpec}
scenarioManagerFactory scenarioManagerFactory = globalSpec.scenarioManagerFactory;
ScenarioManager mainScenarioManager = scenarioManagerFactory.GetMainScenarioManager();



Trouble-Shooting: if the Visualizer (magnifying glass) does not work, please refer to the Appendix 3.2 Trouble-Shooting in Visual Studio Side in the first start guide book to make the Visualizer work.

Clicking the magnifying glass, you will get:



	Name	Value
0	CorporateTaxRate	0.35
1	EarningDistributingInterval	Quarterly
2	FiscalYearEndingMonth	12
3	Run	MyFirstRun
4	SumAssuredPU	1000
5	ThePresentDate	5/31/2012 12:00:00 AM
6	TotalProjectionMonths	4800
7	embeddedValueDiscountRate	0.125
▸8	scenarioManagerFactory	...

And you can click "…" to view nested data.

The following codes read inputs from TurboTerm1Input.xlsb, and process assumptions into global spec, plan spec, and model point spec. They also prepare output table for outputs. These codes are parallel to "Objects" tab in Excel application we introduced in section 3.1 of Quick Start Guide - TurboExcel.

```
string excelFileNameOrDatabaseConnectionString = Utility.Env("TurboRoot") + @"Application\Input\Term\TurboTerm1\TurboTerm1Inputs.xlsx";
DataAccessLayer dataAccessLayer = new DataAccessLayer(excelFileNameOrDatabaseConnectionString);

RunEnum thisRun = RunEnum.MyFirstRun;

GlobalSpec globalSpec = new GlobalSpec(thisRun, dataAccessLayer, "GlobalSpec", "Historical", "NestedNodes", "InterestCurveDefinition", "Vo

ScenarioManagerFactory scenarioManagerFactory = globalSpec.scenarioManagerFactory;
ScenarioManager mainScenarioManager = scenarioManagerFactory.GetMainScenarioManager();

int scenarioNumber = 1;

DataTable planToRun = dataAccessLayer.Read("PlanToRun", "Run = '{0}'".With(thisRun));

DataTable outputTable = DataOutput.CreateTable( new ColumnSpec("Product", typeof(string)), new ColumnSpec("Plan Group", typeof(string)), n
    "Book Profit", "Embedded Value");

foreach (DataRow planRow in planToRun.AsEnumerable())
{
    ProductEnum product = Utility.ToEnum<ProductEnum>(planRow.Field<string>("Product"));
    PlanGroupEnum planGroup = Utility.ToEnum<PlanGroupEnum>(planRow.Field<string>("Plan Group"));
    PlanEnum plan = Utility.ToEnum<PlanEnum>(planRow.Field<string>("Plan"));

    PlanSpec planSpec = new PlanSpec(dataAccessLayer, "Term1", globalSpec, product, planGroup, plan, thisRun);

    DataTable modelPointRows = ModelPointUtility.GetModelPoints(dataAccessLayer, "ModelPointTable", plan);

    IArray<double>[] embeddedValues = new IArray<double>[modelPointRows.Rows.Count];
    IArray<double>[] bookProfits = new IArray<double>[modelPointRows.Rows.Count];

    for (int i = 0; i < modelPointRows.Rows.Count; i++)
    {
        MpSpec mpSpec = new MpSpec(globalSpec, planSpec, modelPointRows.Rows[i], product, planGroup, plan);
```

The following codes calculate embedded values and store results into output\TurboTerm1.csv. These codes are parallel to "Term" tab in Excel application we introduced in chapter 2 of Quick Study Guide - TurboExcel.

```
for (int i = 0; i < modelPointRows.Rows.Count; i++)
{
    MpSpec mpSpec = new MpSpec(globalSpec, planSpec, modelPointRows.Rows[i], product, planGroup, plan);
    Term term = new Term(globalSpec, planSpec, mpSpec, 0, globalSpec.TotalProjectionMonths, mainScenarioManager, scenarioNumber, true);
    term.Calculate();

    embeddedValues[i] = term.embeddedValue;
    bookProfits[i] = term.bookProfit;
}

DataOutput.InsertIntoTable(outputTable, 0, int.MaxValue,
    new ColumnContent("Product", product.ToString()),
    new ColumnContent("Plan Group", planGroup.ToString()),
    new ColumnContent("Plan", plan.ToString()),
    new ColumnContent("Book Profit", bookProfits.Sum()),
    new ColumnContent("Embedded Value", embeddedValues.Sum()));
```

}

3.4 High Performance in Multi-cpu/core Computers

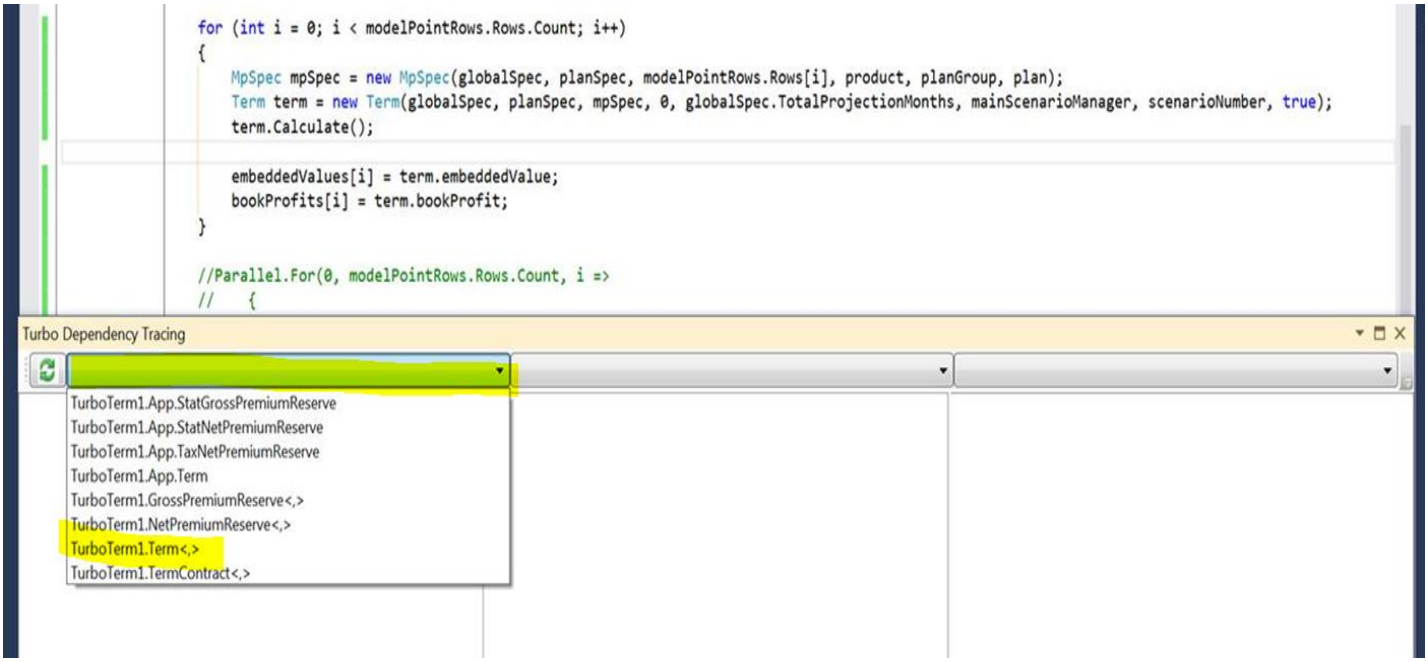
The codes in the above section have only made used of one core of your computer.

Our software is very easy to do parallel computing in multi-cpu/core computers. You only have to change your for loop to be Parallel.For loop.

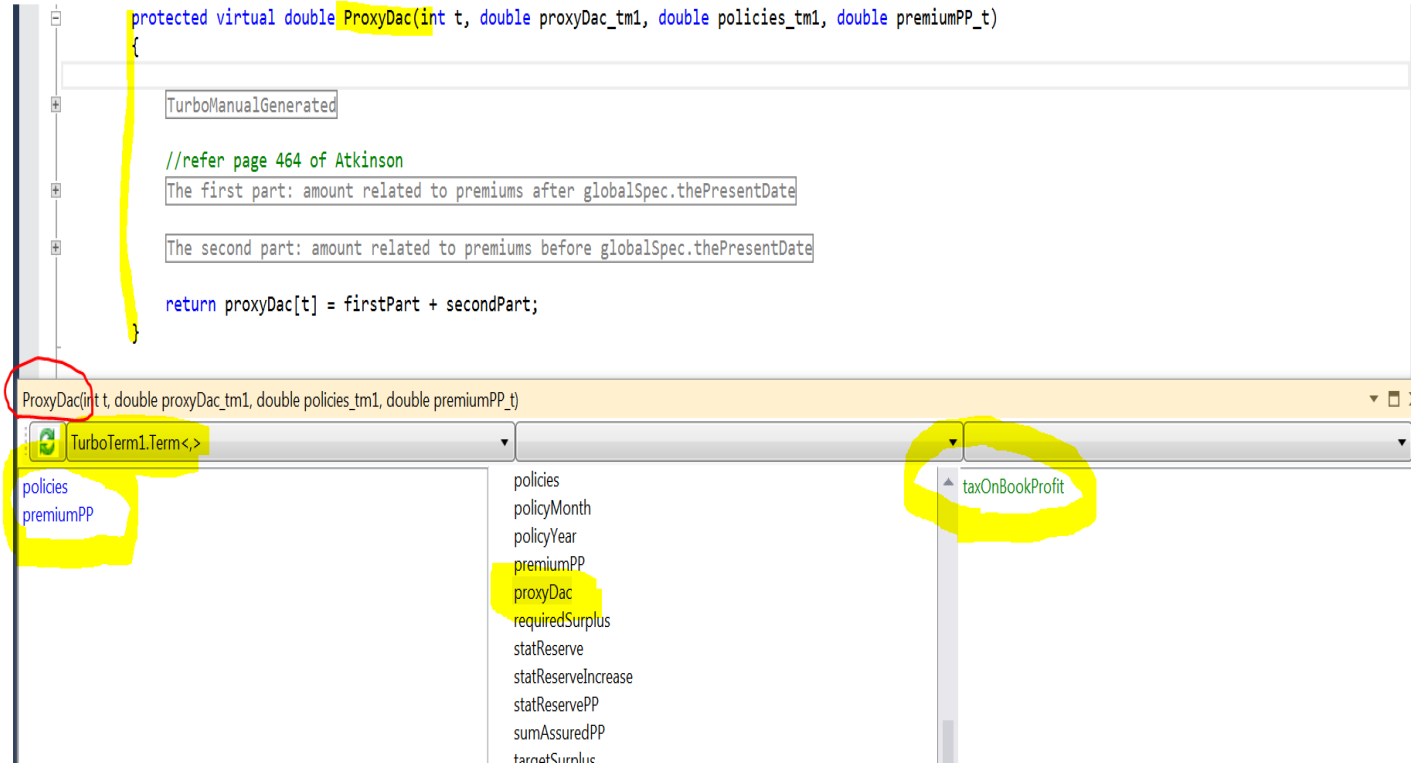
```
//for (int i = 0; i < modelPointRows.Rows.Count; i++)
//{
//    MpSpec mpSpec = new MpSpec(globalSpec, planSpec, modelPointRows.Rows[i], product, planGroup, plan);
//    Term term = new Term(globalSpec, planSpec, mpSpec, 0, globalSpec.TotalProjectionMonths, mainScenarioManager, scenarioNumber, true);
//    term.Calculate();

//    embeddedValues[i] = term.embeddedValue;
//    bookProfits[i] = term.bookProfit;
//}

Parallel.For(0, modelPointRows.Rows.Count, i =>
    {
        MpSpec mpSpec = new MpSpec(globalSpec, planSpec, modelPointRows.Rows[i], product, planGroup, plan);
        Term term = new Term(globalSpec, planSpec, mpSpec, 0, globalSpec.TotalProjectionMonths, mainScenarioManager, scenarioNumber, true);
        term.Calculate();
        embeddedValues[i] = term.embeddedValue;
        bookProfits[i] = term.bookProfit;
    }
);
```

3.5 Dependence Tracing

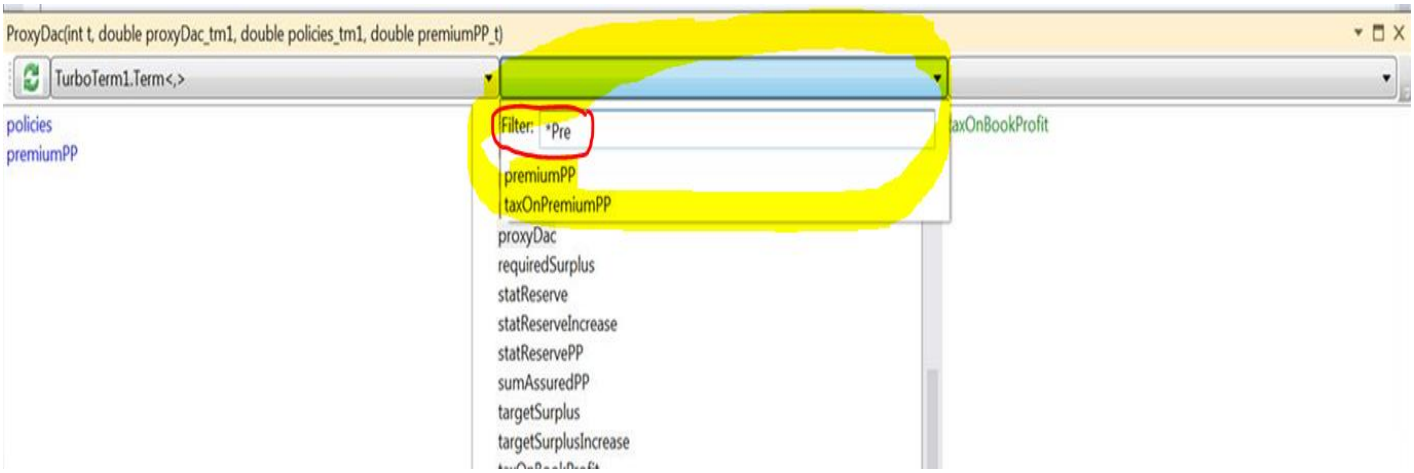
We can also trace precedence and dependence in VS 2012 Professional. Click “Turbo Dependence Tracing” submenu from TurboActuary main menu, dock it on the bottom of VS or float it on your secondary monitor, and you will see:



Choose TurboTerm1.Term<,>, and you will see:



As always, the blue color means precedence and green color means dependence. You can double-click any of these array names (similar as Excel column) or quickly search by:



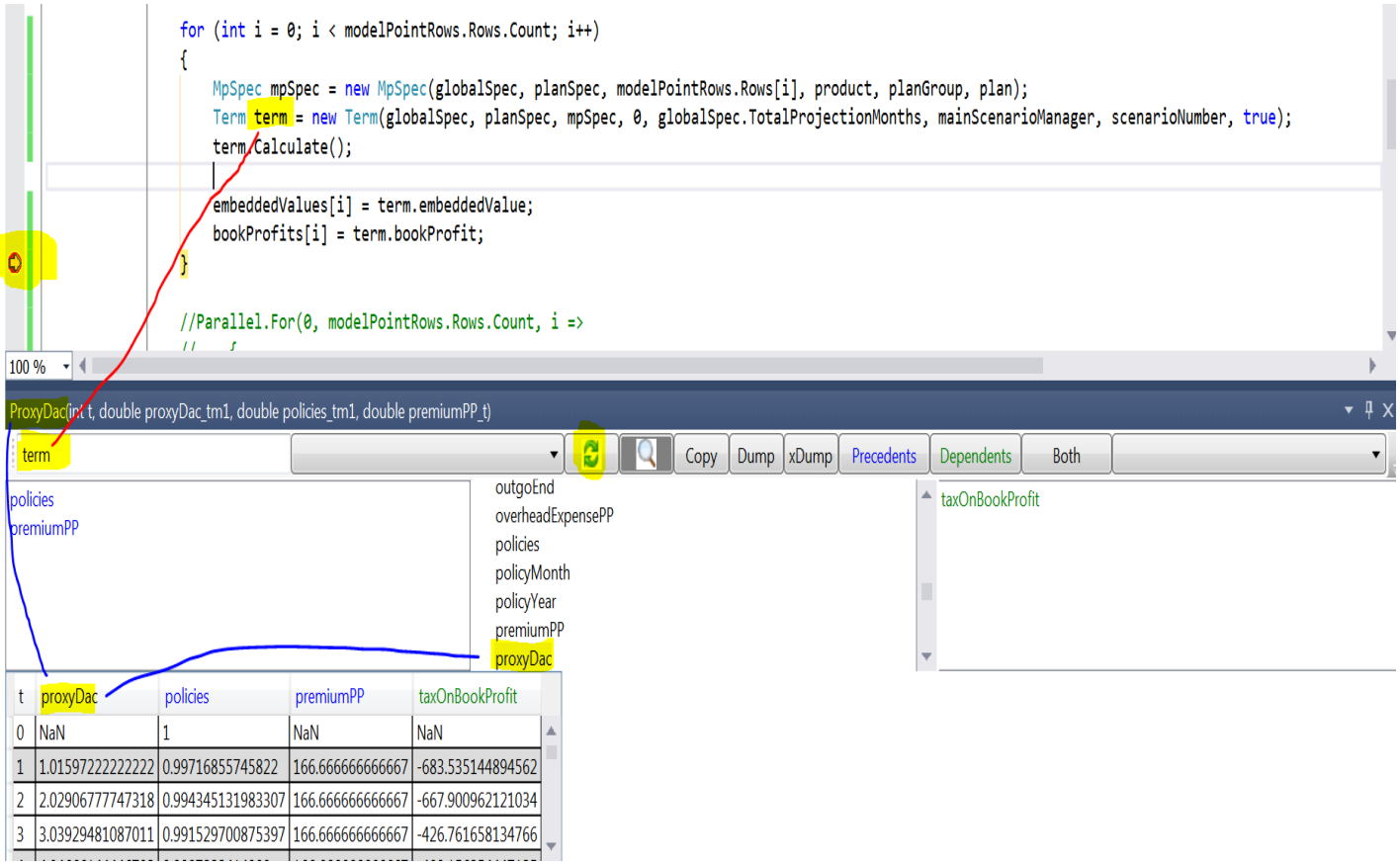
The code window always show the code related to the current center array. The right box in Turbo Dependence Tracing window contains all history arrays that you have visited and let you go back quickly.

3.6 Dependence Debugging

The preceding section help you navigate precedence/dependence relationship between array variables and show their C# codes. This section shows that we can stop at any breakpoint during debugging, and view its data and execution in addition to precedence/dependence relationship.

Set up a breakpoint either outside or inside the TurboTerm1.Term class, then hit F5. The program will run and pause at the breakpoint.

Then click "Turbo Dependence Debugging" submenu in TurboActuary main menu, and type the name of any object having array variables (for example, type "term" if outside TurboTerm1.Term class or "this" if inside TurboTerm1.Term class). Then you click refresh icon and will see:



You can press F10 to step over and F11 to step into, and then click refresh icon to show the most current data. So you can trace and debug your model from beginning to ending of modeling process, and the response time is very quick. Our software is the only one that can provide the nice tool.

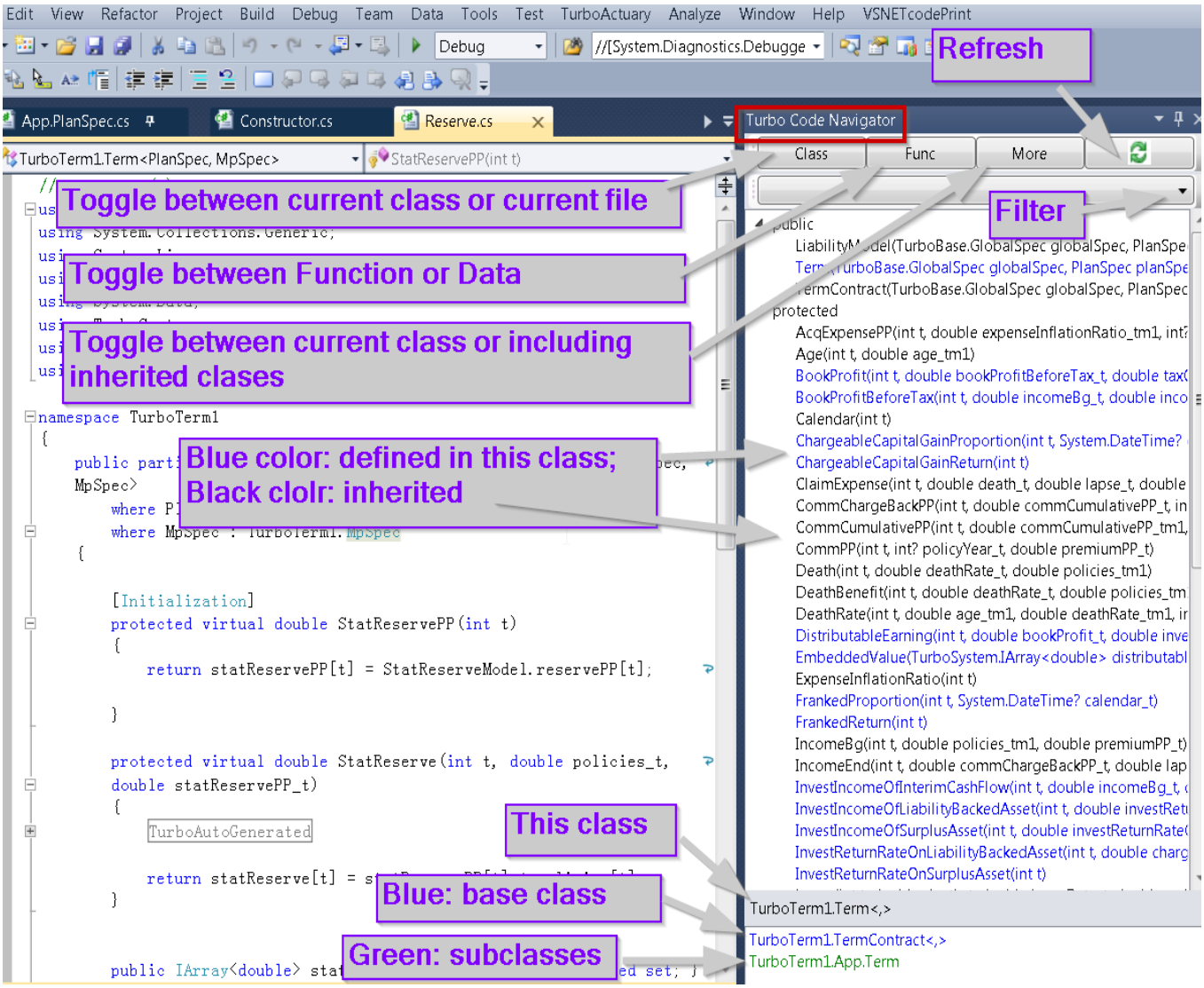
What our competitors provide is to retrieve the results from disk after your run finishes; and you have to output all related data into the disk. They can not debug using the data from memory during execution.

We debug using the data from memory during execution. You do not need to output any data into disk for our debugging purpose.

3.7 Turbo Navigation

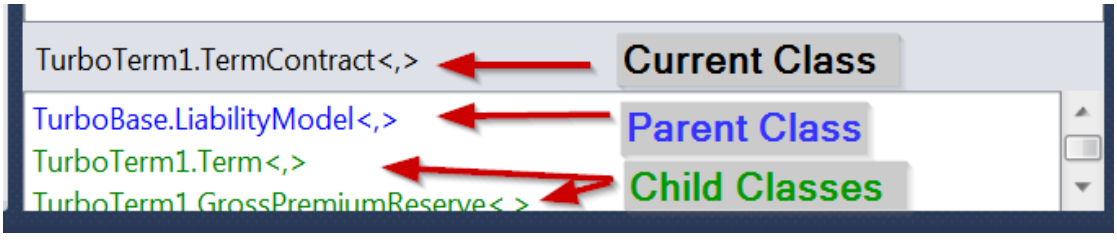
In addition to many navigation tools Visual Studio has provided (introduced in Appendix 2.3), our Turbo Navigation tool will help you quickly find classes, functions, and data fields.

Click "Turbo Navigator" in TurboActuary main menu, you will get:



The refresh button will change the current class to the class in the active edit window. You can use filter to quickly find functions or fields.

The bottom pane in the Navigator shows its parent class and child classes. You can easily see their hierarchy, and walk through them by double-clicking their names.



3.8 Print the Whole Class

Many classes consist of more than one file. The file name is used as category name so you can quickly find your functions based on their categories in both Visual Studio and Excel.

You can color-print all of these files in a class by clicking “Print Current Class” in TurboActuary main menu. This can help you quickly understand the class.

We have more commands under TurboActuary menu, please refer to user manual for details.

Chapter 4: Turbo Intelligent

In this chapter, we use the original TurboTerm1.App.sln as example.

In the next chapter, we show how to customize TurboTerm1.App for your business.

You might cross-read the two chapters together for deeper understanding.

4.1 Highlight

You can double-click any word and it will highlight all words related to your selection no matter it is in the same file or not. (Nobody except us can do highlights across files!)

```
public IArray<double> maintExpensePP { get; protected set; }
protected virtual double MaintExpensePP(int t, double expenseInflationRatio_t, double premiumPP_t, double sumAssuredPP_tm1)
{
    TurboAutoGenerated

    return maintExpensePP[t] = planSpec.FixedMaintExpensePP * expenseInflationRatio[t]
        + planSpec.MaintExpensePremiumPercent * premiumPP[t]
        + planSpec.MaintExpenseBenefitPercent * sumAssuredPP[t - 1];

}

public IArray<double> overheadExpensePP { get; protected set; }
protected virtual double OverheadExpensePP(int t, double expenseInflationRatio_t, double sumAssuredPP_tm1)
{
    TurboAutoGenerated


    return overheadExpensePP[t] = planSpec.FixedOverheadExpensePP * expenseInflationRatio[t]
        + planSpec.OverheadExpenseBenefitPercent * sumAssuredPP[t - 1];
}
```

You can double-click any space (outside of word) to underscore all arrays.

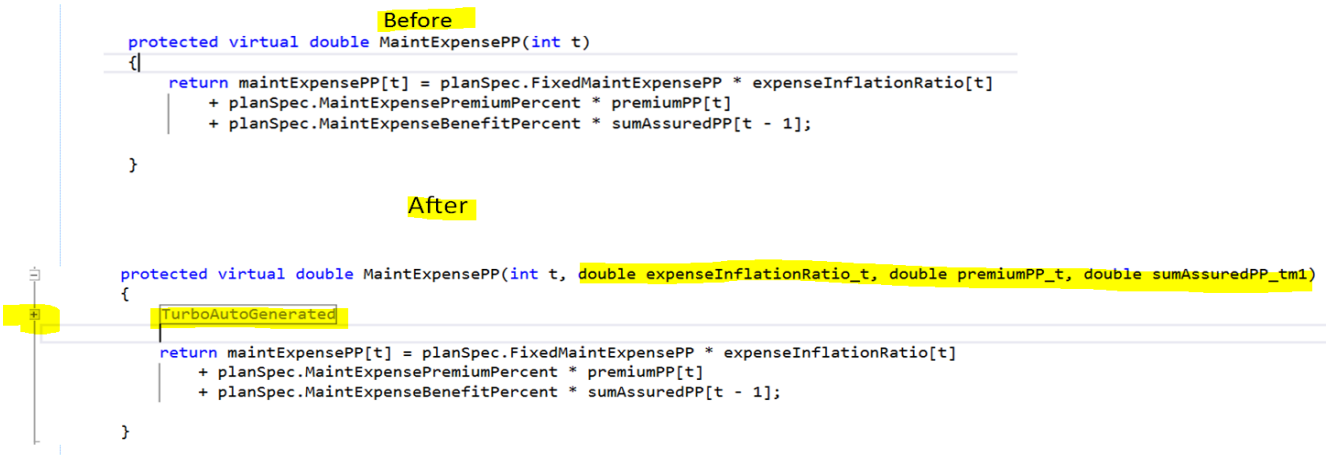
```
public IArray<double> premiumPP { get; protected set; }
protected virtual double PremiumPP(int t, int? policyMonth_t, int? policyYear_t)
{
    TurboAutoGenerated

    if (mpSpec.PremiumInterval == IncidentIntervalEnum.SinglePremium)
    {
        if (policyYear[t].Value == 1 && policyMonth[t].Value == 1)
            return premiumPP[t] = mpSpec.PremiumPP;
        else
            return premiumPP[t] = 0.0;
    }

    if (policyYear[t].Value > mpSpec.PremiumYearTerm)
        return premiumPP[t] = 0.0;

    if ((policyMonth[t].Value - 1) % (int)mpSpec.PremiumInterval != 0)
        return premiumPP[t] = 0.0;

    return premiumPP[t] = mpSpec.PremiumPP * (int)mpSpec.PremiumInterval / 12.0;
}
```

You can put a cursor on any symbol and it will automatically highlight all references in the current document. You can navigate through the highlights by using ctrl+shift+down array (forward) or ctrl+shift+up array (reverse).

```
private void SomeCoolMethod()
{
    // blah
}

private void A()
{
    SomeCoolMethod();
}

private void B()
{
    SomeCoolMethod();
}

private void C()
{
    SomeCoolMethod();
}
```

4.2 Auto-Generate Parameters

You can auto-generate parameters for your model functions by clicking "Generating Parameter for this Method" in TurboActuary main menu as shown below:



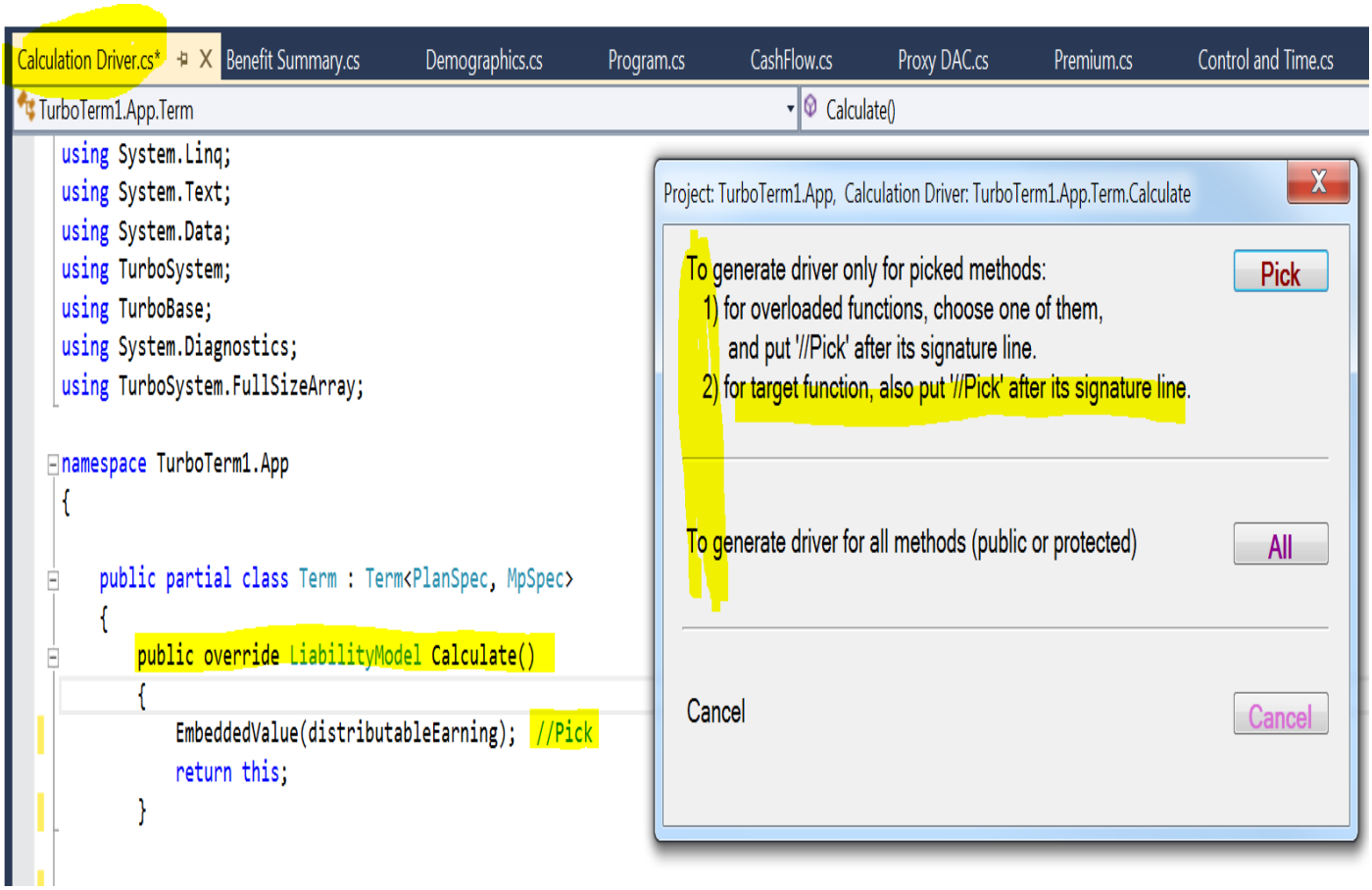
You can also generate parameters for all functions in a class by just a click on "Generating Parameter for this Class" in TurboActuary main menu.

4.3 Auto-Generate Calculation Driver

Our software will automatically assemble small functions to be calculation driver.

Let the 'Calculate' function as shown below. That is we let EmbeddedValue as our targeting array.

Then click "Auto generate calculation driver" in TurboActuary main menu, and then click 'Pick' button.



Trouble-Shooting: please use TurboBase project as reference rather than use TurboBase.dll as reference. Otherwise, it may complain some functions/arrays (which are defined in TurboBase) not found.

It will generate the 'Calculate' function as:

```
for (t = theFirstMonth; t <= ProjectionToMonth; t++)
{
    //layer 1
    Age(t, age[t - 1]);
    Calendar(t);
    ChargeableCapitalGainReturn(t);
    ExpenseInflationRatio(t);
    FrankedReturn(t);
    InvestReturnRateOnSurplusAsset(t);
    PolicyMonth(t);
    PolicyYear(t);
    StatReservePP(t);
    SumAssuredPP(t);
    TaxReservePP(t);
    UnchargeableCapitalGainReturn(t);
    UnfrankedReturn(t);

    //layer 2
    ChargeableCapitalGainProportion(t, calendar[t]);
    DeathRate(t, age[t - 1], deathRate[t - 1], policyMonth[t], policyYear[t]);  // Has [NoInitialization]
    FrankedProportion(t, calendar[t]);
    InvestIncomeOfSurplusAsset(t, investReturnRateOnSurplusAsset[t], targetSurplus[t - 1]);
    LapseRate(t, policyYear[t]);
    OverheadExpensePP(t, expenseInflationRatio[t], sumAssuredPP[t - 1]);
    PremiumPP(t, policyMonth[t], policyYear[t]);
    UnchargeableCapitalGainProportion(t, calendar[t]);
    UnfrankedProportion(t, calendar[t]);

    //layer 3
    AcqExpensePP(t, expenseInflationRatio[t - 1], policyMonth[t], policyYear[t], premiumPP[t]);
    CommPP(t, policyYear[t], premiumPP[t]);  // Has [NoInitialization]
    Death(t, deathRate[t], policies[t - 1]);  // Has [NoInitialization]
    DeathBenefit(t, deathRate[t], policies[t - 1], sumAssuredPP[t - 1]);
```

Delete the function ‘ArrayAllocation’.

Put your cursor in ‘Calculate’ function

Then click “Auto Generate IArray Allocation Code From this Calculation Driver”; rename the newly generated function to be ‘ArrayAllocation’; and you will get:

```
private void ArrayAllocations()
{

    #region Array Allocations AutoGenerated By TurboIntelliSense

    int theFirstMonth = ProjectionFromMonth + 1;

    #region Benefit Summary
    deathBenefit = new SoloSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    sumAssuredPP = new DuetSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    #endregion Benefit Summary

    #region Capital
    requiredSurplus = new SoloSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    targetSurplus = new DuetSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    targetSurplusIncrease = new SoloSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    #endregion Capital

    CashFlow

    #region Control and Time
    calendar = new SoloSlidingArray<System.DateTime?>(ProjectionFromMonth, ProjectionToMonth);
    policyMonth = new SoloSlidingArray<int?>(ProjectionFromMonth, ProjectionToMonth);
    policyYear = new SoloSlidingArray<int?>(ProjectionFromMonth, ProjectionToMonth);
    #endregion Control and Time

    #region Demographics
    age = new DuetSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    death = new SoloSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    deathRate = new DuetSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    lapse = new SoloSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    lapseRate = new SoloSlidingDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    policies = new SimpleDoubleArray(ProjectionFromMonth, ProjectionToMonth);
    #endregion Demographics
```

The Simple Array will allocate memory for all projection months. However, the Solo Sliding Array will allocate only one element and keep rolling with your projection; the Duet Sliding Array will allocate only two elements and keep rolling with your projection.

Our intelligent tool will automatically determine which kind of array it should use for best memory usage.

Thus our software is extremely memory efficient when you choose to use namespace `TurboSystem.MemoryEfficientArray` instead of `TurboSystem.FullSizeArray`.

You can see that our calculation in the driver is always starting from the beginning of projection, then to t+1 and until policy matures. So you can trace your calculation very easily.

Other vendors may execute their programs out of time order, thus hard to trace and understand their executions.

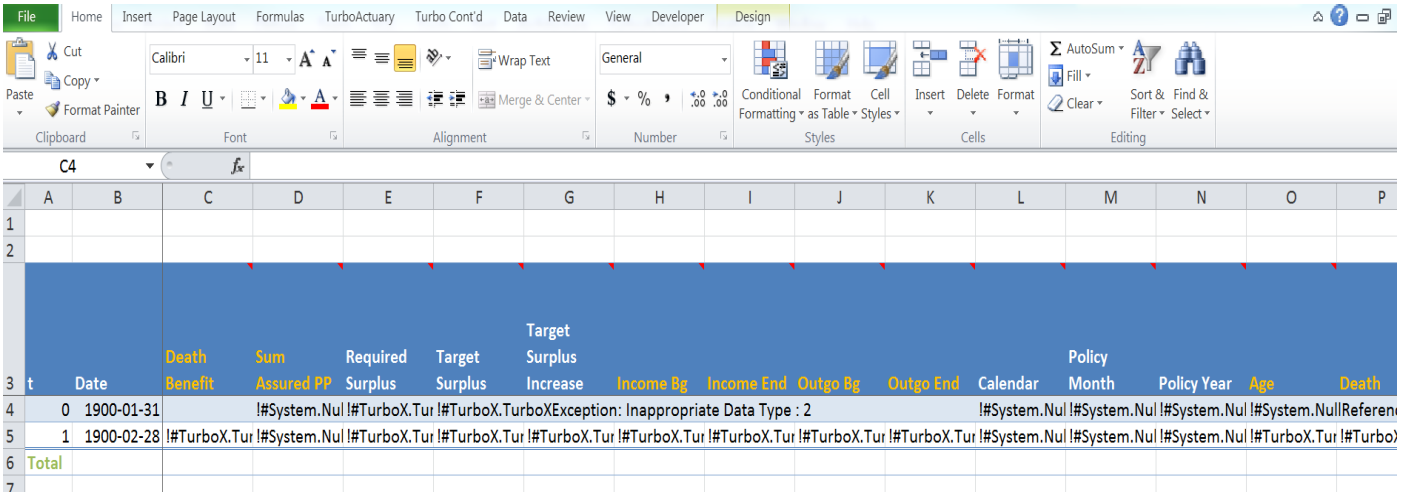
4.4 Auto-Generate Excel Model

Other vendors use Excel to retrieve the calculated results, but we generate the real actuarial Excel models automatically for you!

Create a spreadsheet with "Objects" tab (what you need to do is to copy from example and make slight change since they almost have the same formulas except different object names).

For simplicity, at this time you can create a copy of TurboTerm1.App.xlsb, and delete other three tabs except "Objects" tab.

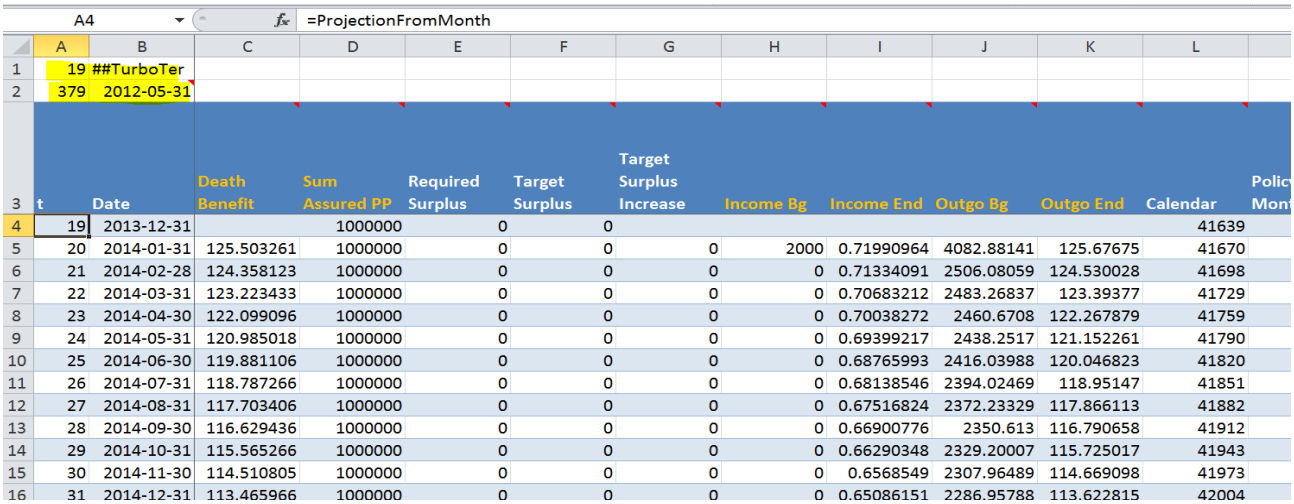
Then Go to your calculation driver in Visual Studio, and click "Auto Generate Excel Formula From This Calculation Driver" in TurboActuary main menu, and click OK, and xCopy, you will get a new worksheet in your Excel workbook:



t	Date	Death Benefit	Sum Assured PP	Required Surplus	Target Surplus	Target Surplus Increase	Income Bg	Income End	Outgo Bg	Outgo End	Calendar	Policy Month	Policy Year	Age	Death
0	1900-01-31		!#System.Nul	!#TurboX.Tur	!#TurboX.Tur	TurboXException: Inappropriate Data Type : 2					!#System.Nul	!#System.Nul	!#System.Nul	!#System.Nul	Referen
1	1900-02-28	!#TurboX.Tur	!#System.Nul	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#System.Nul	!#System.Nul	!#System.Nul	!#TurboX.Tur	!#Turbo
Total															

Now put some assumptions into the new sheet: cell A1: = Models[Projection From Month], cell A2: = Models[Projection To Month], cell B1: = Models[Term], and cell B2 = GlobalObjects[The Present Date]. Then in cell B1, click "Object Format" in "Turbo Cont'd" ribbon menu to format the cell.

Then activate any cell in the Excel table (say A4), and click "Auto R" in TurboActuary ribbon menu, you will automatically generate your Excel model successfully:



t	Date	Death Benefit	Sum Assured PP	Required Surplus	Target Surplus	Target Surplus Increase	Income Bg	Income End	Outgo Bg	Outgo End	Calendar	Policy Mont
19	2013-12-31		1000000	0	0						41639	
20	2014-01-31	125.503261	1000000	0	0	0	2000	0.71990964	4082.88141	125.67675	41670	
21	2014-02-28	124.358123	1000000	0	0	0	0	0.71334091	2506.08059	124.530028	41698	
22	2014-03-31	123.223433	1000000	0	0	0	0	0.70683212	2483.26837	123.39377	41729	
23	2014-04-30	122.099096	1000000	0	0	0	0	0.70038272	2460.6708	122.267879	41759	
24	2014-05-31	120.985018	1000000	0	0	0	0	0.69399217	2438.2517	121.152261	41790	
25	2014-06-30	119.881106	1000000	0	0	0	0	0.68765993	2416.03988	120.046823	41820	
26	2014-07-31	118.787266	1000000	0	0	0	0	0.68138546	2394.02469	118.95147	41851	
27	2014-08-31	117.703406	1000000	0	0	0	0	0.67516824	2372.23329	117.866113	41882	
28	2014-09-30	116.629436	1000000	0	0	0	0	0.66900776	2350.613	116.790658	41912	
29	2014-10-31	115.565266	1000000	0	0	0	0	0.66290348	2329.20007	115.725017	41943	
30	2014-11-30	114.510805	1000000	0	0	0	0	0.6568549	2307.96489	114.669098	41973	
31	2014-12-31	113.465966	1000000	0	0	0	0	0.65086151	2286.95788	113.622815	42004	

This is a full-fledged model. As we have shown, it is well-organized and extremely traceable. Inside and out, you can play your actuarial models in Excel at any modeling level,

automatically, instantly, interactively and with business intelligence without reinventing a single wheel. (Write Once, Use Everywhere!)

Chapter 5: Customize the Model for Your Business

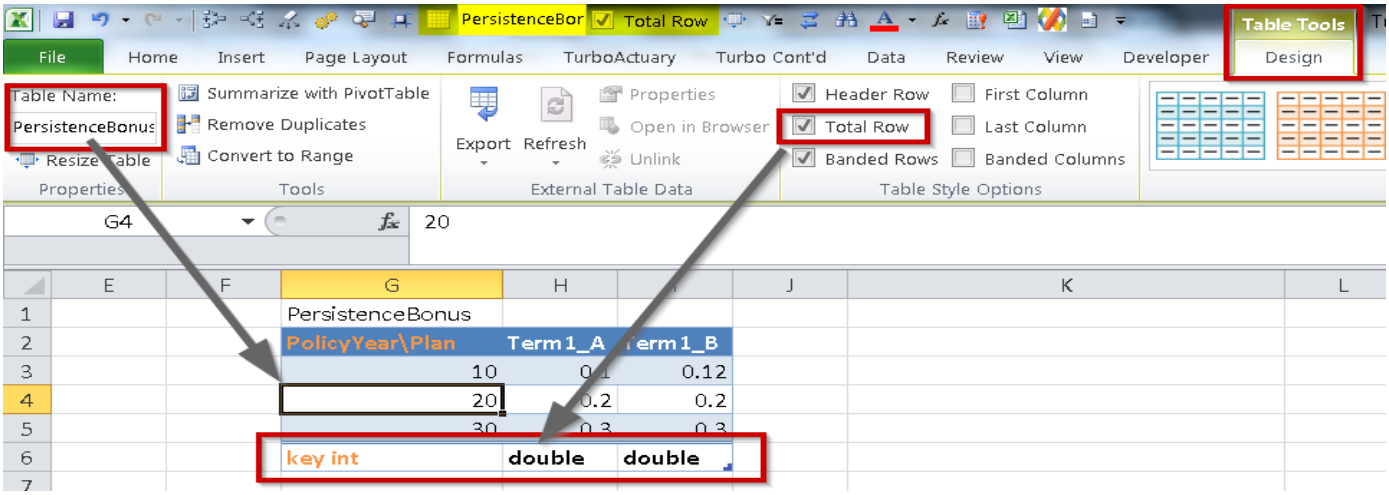
As we have said before, you can build model incrementally. Your new model will inherits and reuses your existed base model, but keep the base model intact.

Please refer to the model TurboTerm1.RPR.App for an example. It inherits and reuses the base model TurboTerm1.App with the base model intact. The new model TurboTerm1.RPR.App builds risk premium reinsurance logic above the base model.

Here we show a simple example to add new business logics directly into TurboTerm1.App model instead of extending the base model incrementally.

5.1 Add a New Assumption Table into Input File

Open C:\TurboActuary3.0\Application\Input\Term\TurboTerm1\TurboTerm1Inputs.xlsb file. And create an Excel assumption table with name "PersistenceBonus" as followings:



Please refer to Appendix 1.2 of Quick Start Guide – TurboExcel for how to create an Excel table for assumption use.

5.2 Read in the Assumption table as a Pivot Table into PlanSpec

Go to App.PlanSpec.cs in the project TurboTerm1.App of the solution TurboTerm1.App. Add the highlight lines.

```
public PlanSpec(DataAccessLayer dataAccessLayer, string planSpecName, GlobalSpec globalSpec, ProductEnum product,
    PlanGroupEnum planGroup, PlanEnum plan, RunEnum thisRun = default(RunEnum))
    : base(planSpecName, globalSpec, product, planGroup, plan)
{
    #region TableSet
    this.Tables.Add("Plan", dataAccessLayer.Read("Plan", "Run = '{0}'".With(thisRun), null)); //Plan Table

    this.InsuranceTableEngines.Add(dataAccessLayer, "MortalityTables", globalSpec.Run, plan); //Mortality

    this.PivotTablesIP.Add("PersistenceBonus", dataAccessLayer.Read<int, PlanEnum>
        ("PersistenceBonus", IfNoExactMatchEnum.UseSpecified, IfNoExactMatchEnum.ThrowException, 0.0));

    this.PivotTablesIP.Add("SumAssuredIndex", dataAccessLayer.Read<int, PlanEnum>
        ("SumAssuredIndex", IfNoExactMatchEnum.UsePriorOne, IfNoExactMatchEnum.ThrowException));
```

This line of code will read the PersistencceBonus table from Excel input file for later use.

The 'I' in the PivotTableIP means that row is integer type (here it is a policy year); the 'P' in the PivotTableIP means that column is PlanEnum type (here it is Term1_A or Term1_B).

The 'IfNoExactMatchEnum.UseSpecified' means that it returns the specified value 0 if no exactly matched row is found; the 'IfNoExactMatchEnum.ThrowException' means that it will throw an exception if no exactly matched column is found.

5.3 Add Persistence Bonus Business Logic

Since it is insurer's expense, we go to Expense.cs file of TermContract of TurboTerm1 project, and add the following lines:

```
public IArray<double> premiumReimbursePP { get; protected set; }
[NoInitialization]
protected virtual double PremiumReimbursePP(int t)
{
    var pivotTable = planSpec.PivotTablesIP["PersistenceBonus"];
    double percent = pivotTable[policyYear[t].Value, planSpec.Plan];

    return premiumReimbursePP[t] = premiumPP[t] * percent;
}
```

Please note that we put [NoInitialization] before the function since the function need not to be executed for initial period; Otherwise, we can put [Initialization] before the function.

Now put the cursor in the body of the above function, and click "Generate Parameter for this method" under TurboActuary menu. It will automatically generate parameters and debug assertions for these parameters.

```
public IArray<double> premiumReimbursePP { get; protected set; }
[NoInitialization]
protected virtual double PremiumReimbursePP(int t, int? policyYear_t, double premiumPP_t)
{
    #region TurboAutoGenerated
    TurboDebug.Assert(policyYear, policyYear_t, t);
    TurboDebug.Assert(premiumPP, premiumPP_t, t);
    #endregion TurboAutoGenerated

    var pivotTable = planSpec.PivotTablesIP["PersistenceBonus"];
    double percent = pivotTable[policyYear[t].Value, planSpec.Plan];

    return premiumReimbursePP[t] = premiumPP[t] * percent;
}
```

5.4 Update Beginning Cash Flow Function

In the CashFlow.cs of project TurboTerm1, we are going to update the OutgoBg function so that it incorporates the premiumReimbursePP item, and then redo the "Generate Parameter for this method" under TurboActuary menu.

```
public IArray<double> outgoBg { get; protected set; }
protected virtual double OutgoBg(int t, double acqExpensePP_t, double commPP_t, double maintExpensePP_t,
    double overheadExpensePP_t, double policies_tm1, double premiumReimbursePP_t, double taxOnPremiumPP_t)
{
    #region TurboAutoGenerated
    TurboDebug.Assert(acqExpensePP, acqExpensePP_t, t);
    TurboDebug.Assert(commPP, commPP_t, t);
    TurboDebug.Assert(maintExpensePP, maintExpensePP_t, t);
    TurboDebug.Assert(overheadExpensePP, overheadExpensePP_t, t);
    TurboDebug.Assert(policies, policies_tm1, t - 1);
    TurboDebug.Assert(premiumReimbursePP, premiumReimbursePP_t, t);
    TurboDebug.Assert(taxOnPremiumPP, taxOnPremiumPP_t, t);
    #endregion TurboAutoGenerated

    return outgoBg[t] = (taxOnPremiumPP[t] + commPP[t] + acqExpensePP[t]
        + maintExpensePP[t] + overheadExpensePP[t] + premiumReimbursePP[t]) * policies[t - 1];

}
```

5.5 Generate the Calculation Driver

Go to 'Calculation Driver.cs' of folder term of project TurboTerm1.App.

1) Delete the function ArrayAllocation.
2) Put the cursor inside the function Calculate, and click the "Auto Generate Calculation Driver Code" submenu. Choose 'Pick' to generate the calculation driver for the picked column EmbeddedValue.

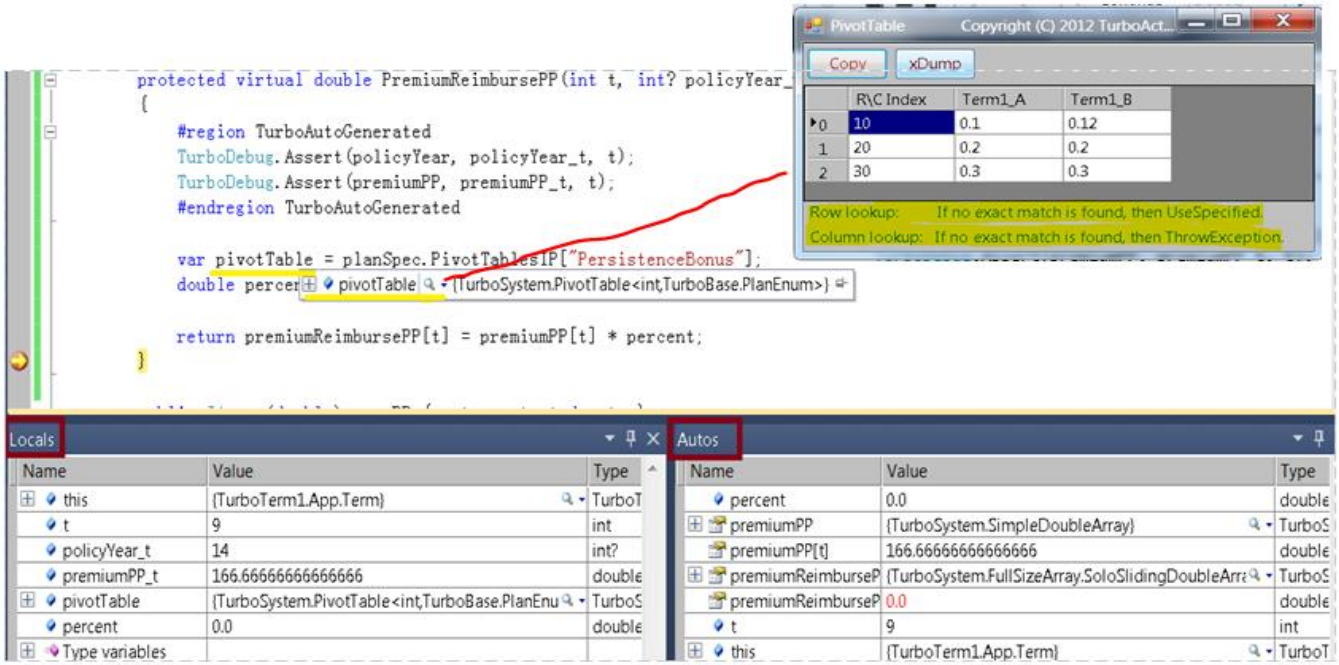
```
EmbeddedValue(distributableEarning);  //Pick
return this;
```

3) Put the cursor inside the function Calculate, and click the "Auto Generate IArray Allocation Code" submenu. Please rename the generated function as 'ArrayAllocations'.

Now please repeat the above three steps for the 'Calculation Driver.cs' of folder StatGrossPremiumReserve of project TurboTerm1.App.

5.6 Recompiling and Running in VS

Rebuild solution in Debug configuration, hit F5 to run it. You can debug it by setting breakponts. Please refer to section 3.3 for the execution tracing.



5.7 Generate Excel Model

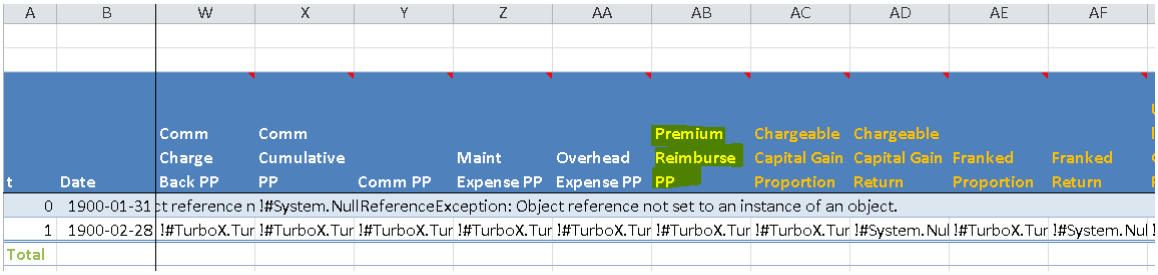
Copy C:\TurboActuary3.0\Application\Excel - Liability\Term\TurboTerm1.App.xlsb to be TurboTerm1.App2.xlsb.

Delete other tabs except the Objects tab, and keep only this workbook open.

Follow the section 4.4 to auto-generate your full-fledged Excel model. Or:

Go to '<u>Calculation Driver.cs</u>' of folder <u>term</u> of project <u>TurboTerm1.App</u>. Put the cursor inside the function <u>Calculate</u>, and click the "Auto Generate Excel Formula From Calculation Driver" submenu. Click OK, then xCopy.

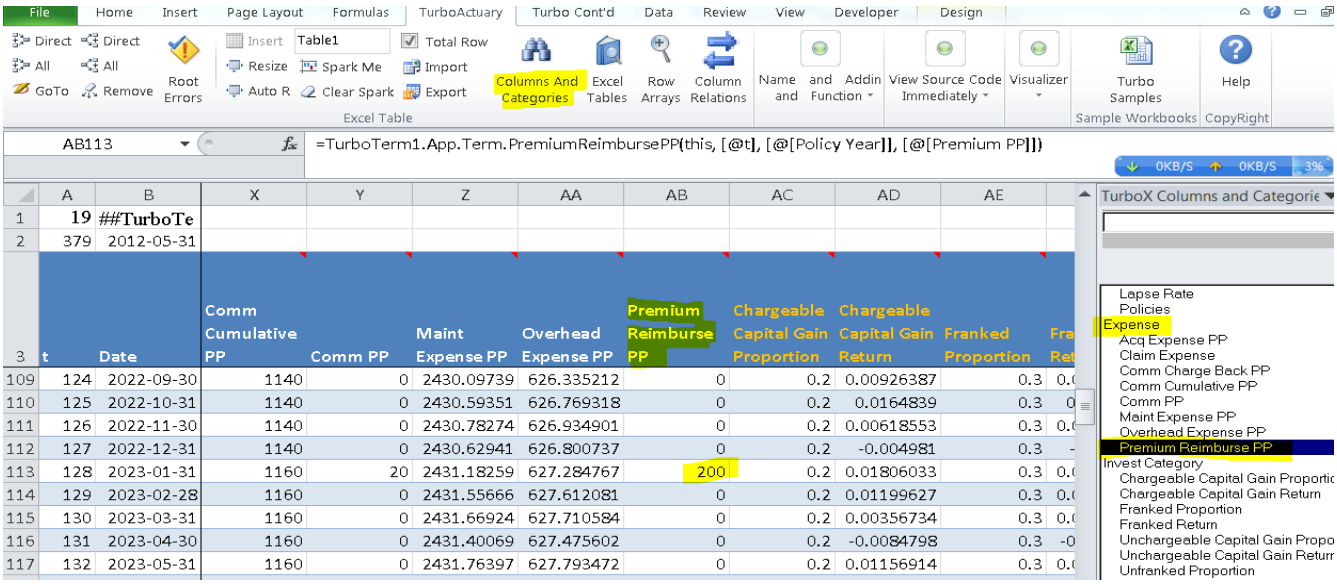
Now you go back to the opened Excel workbook, you will see it auto-generates a new tab with Excel table.



t	Date	Comm Charge Back PP	Comm Cumulative PP	Comm PP	Maint Expense PP	Overhead Expense PP	Premium Reimburse PP	Chargeable Capital Gain Proportion	Chargeable Capital Gain Return	Franked Proportion	Franked Return
0	1900-01-31	ct reference n !#System.NullReferenceException: Object reference not set to an instance of an object.									
1	1900-02-28	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#TurboX.Tur	!#System.Nul	!#TurboX.Tur	!#System.Nul
Total											

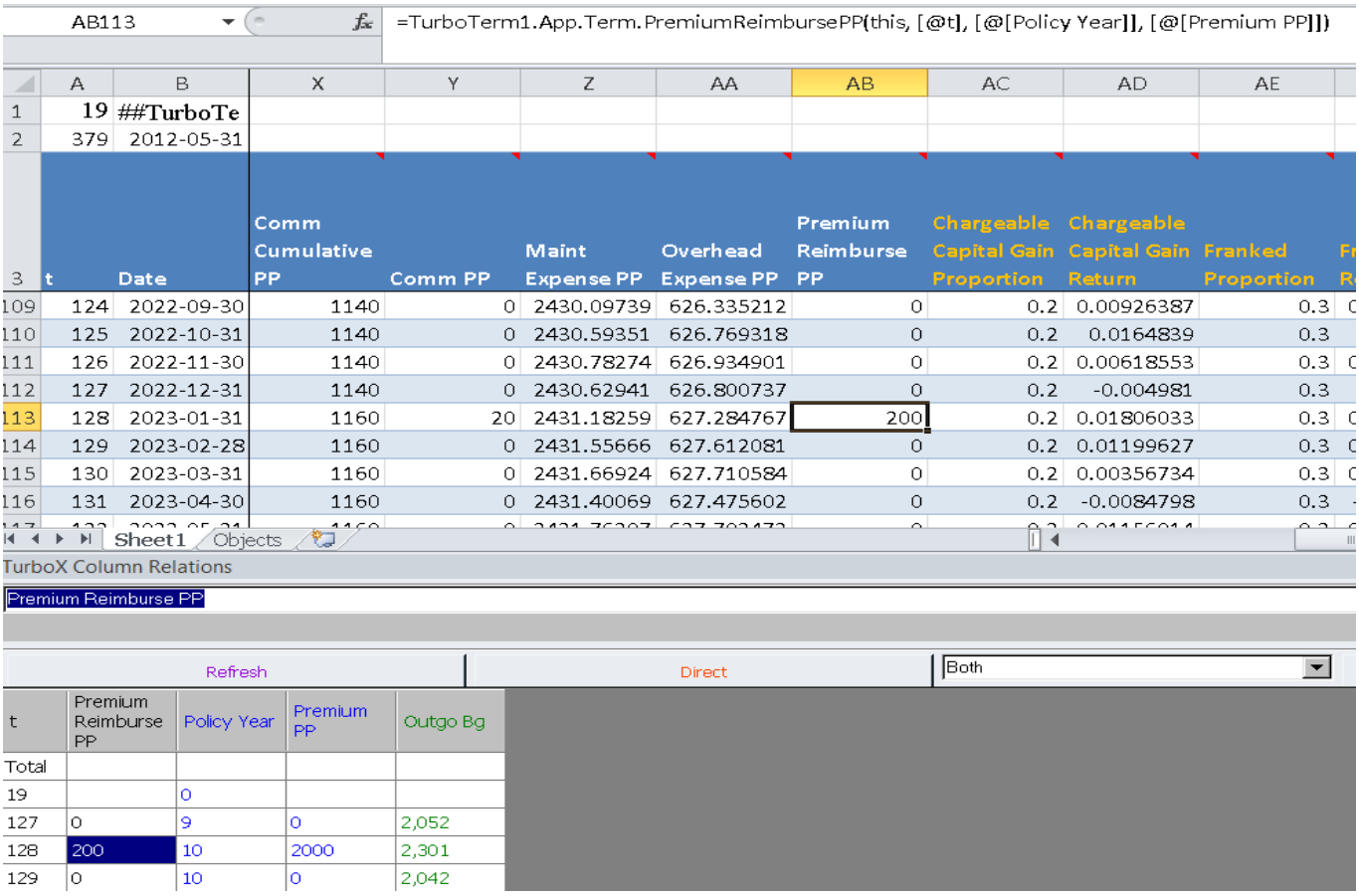
Now put some assumptions into the new sheet: A1 = Models[Projection From Month], A2 =Models[Projection To Month], B1 = Models[Term], B2 = GlobalObjects[The Present Date]. Then in cell B1, click "Object Format" in "Turbo Cont'd" ribbon menu to format the cell.

Then activate any cell in the Excel table (say A4), and click "Auto R" in TurboActuary ribbon menu, you will automatically generate your Excel model successfully. To quickly find the 'PremiumReimbursePP', you can click the 'Columns and Categories' to locate it.



t	Date	Comm Cumulative PP	Comm PP	Maint Expense PP	Overhead Expense PP	Premium Reimburse PP	Chargeable Capital Gain Proportion	Chargeable Capital Gain Return	Franked Proportion	Fra Ret
19	##TurboTe									
379	2012-05-31									
124	2022-09-30	1140	0	2430.09739	626.335212	0	0.2	0.00926387	0.3	0.0
125	2022-10-31	1140	0	2430.59351	626.769318	0	0.2	0.0164839	0.3	0
126	2022-11-30	1140	0	2430.78274	626.934901	0	0.2	0.00618553	0.3	0.0
127	2022-12-31	1140	0	2430.62941	626.800737	0	0.2	-0.004981	0.3	-
128	2023-01-31	1160	20	2431.18259	627.284767	200	0.2	0.01806033	0.3	0.0
129	2023-02-28	1160	0	2431.55666	627.612081	0	0.2	0.01199627	0.3	0.0
130	2023-03-31	1160	0	2431.66924	627.710584	0	0.2	0.00356734	0.3	0.0
131	2023-04-30	1160	0	2431.40069	627.475602	0	0.2	-0.0084798	0.3	-0
132	2023-05-31	1160	0	2431.76397	627.793472	0	0.2	0.01156914	0.3	0.0

You can investigate its dependence relationship by clicking '<u>Column Relations</u>' with active cell being AB113.



AB113 =TurboTerm1.App.Term.PremiumReimbursePP(this, [@t], [@[Policy Year]], [@[Premium PP]])

	A	B	X	Y	Z	AA	AB	AC	AD	AE
1	19	##TurboTe								
2	379	2012-05-31								
3	t	Date	Comm Cumulative PP	Comm PP	Maint Expense PP	Overhead Expense PP	Premium Reimburse PP	Chargeable Capital Gain Proportion	Chargeable Capital Gain Return	Franked Proportion
109	124	2022-09-30	1140	0	2430.09739	626.335212	0	0.2	0.00926387	0.3
110	125	2022-10-31	1140	0	2430.59351	626.769318	0	0.2	0.0164839	0.3
111	126	2022-11-30	1140	0	2430.78274	626.934901	0	0.2	0.00618553	0.3
112	127	2022-12-31	1140	0	2430.62941	626.800737	0	0.2	-0.004981	0.3
113	128	2023-01-31	1160	20	2431.18259	627.284767	200	0.2	0.01806033	0.3
114	129	2023-02-28	1160	0	2431.55666	627.612081	0	0.2	0.01199627	0.3
115	130	2023-03-31	1160	0	2431.66924	627.710584	0	0.2	0.00356734	0.3
116	131	2023-04-30	1160	0	2431.40069	627.475602	0	0.2	-0.0084798	0.3

Sheet1 Objects

TurboX Column Relations

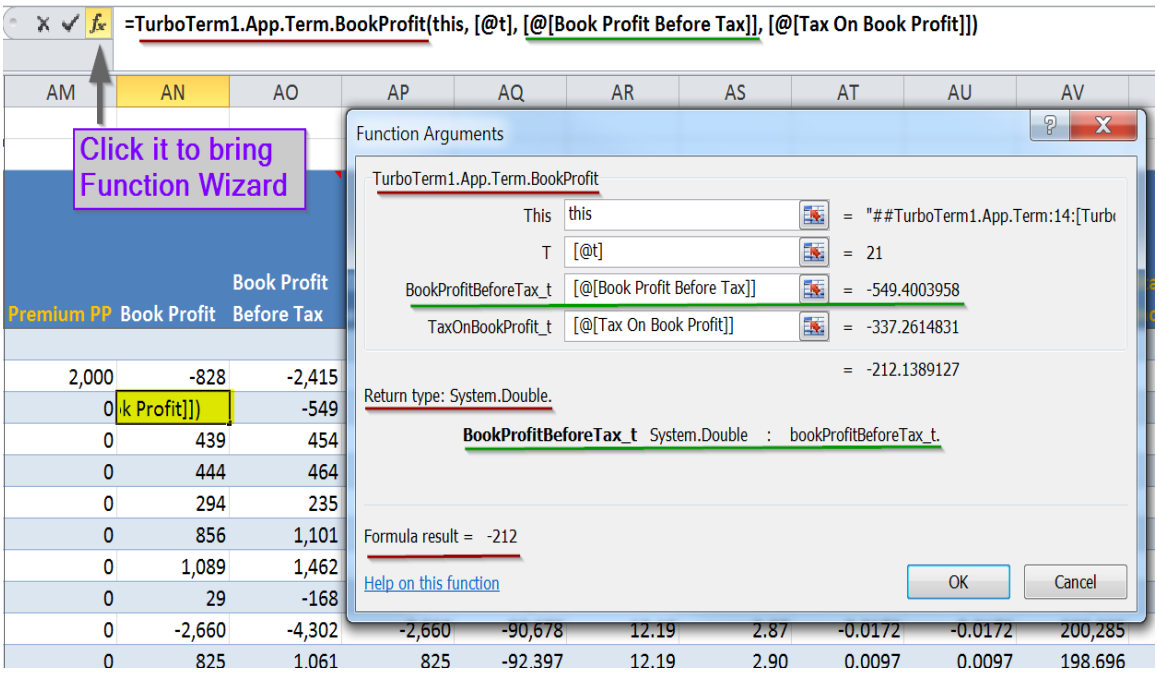
Premium Reimburse PP

Refresh Direct Both

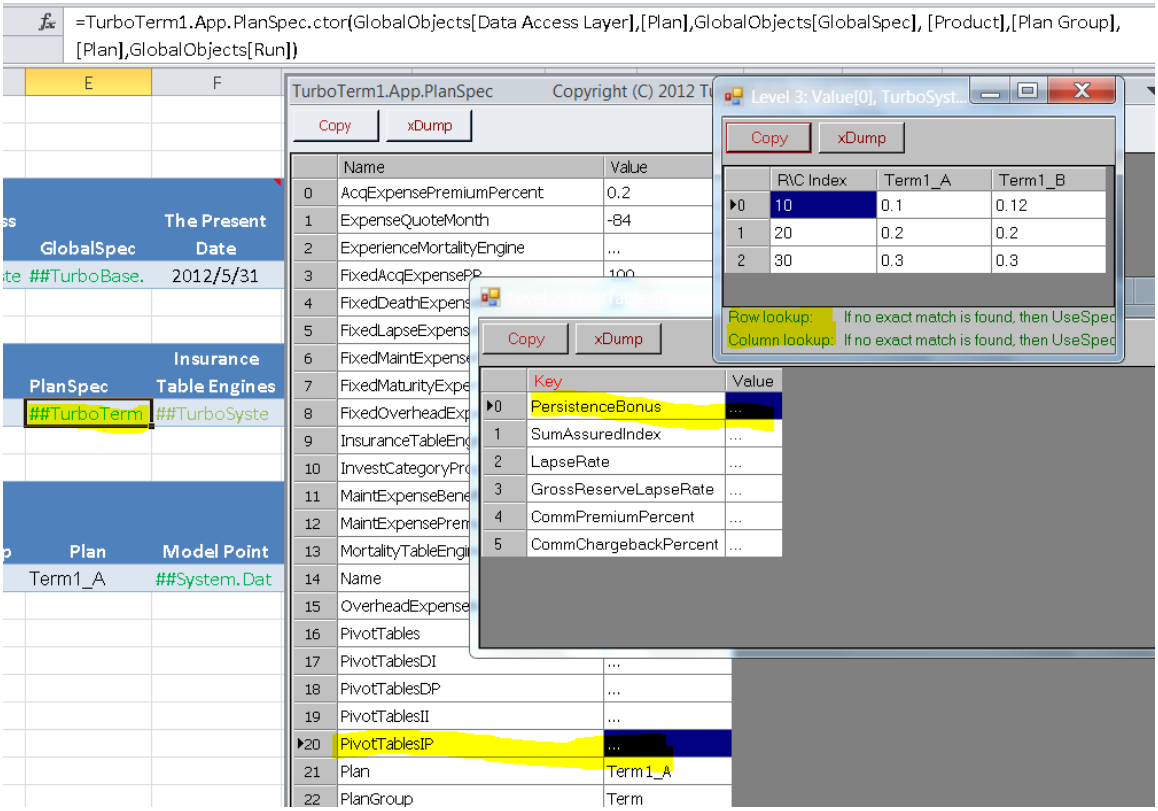
t	Premium Reimburse PP	Policy Year	Premium PP	Outgo Bg
Total				
19		0		
127	0	9	0	2,052
128	200	10	2000	2,301
129	0	10	0	2,042

You can see the PremiumReimbursePP has pre-dependence columns 'Policy Year' and 'Premium PP', and post-dependence 'Outgo Bg'.

Excel function wizard can tell you lots of useful information about your function calling.



To view model assumptions, you can go to Objects tab, activate cell E9, and click the 'Object' submenu. You will see by clicking '…' for all assumption details:



5.8 View Excel Model Formula C# Code

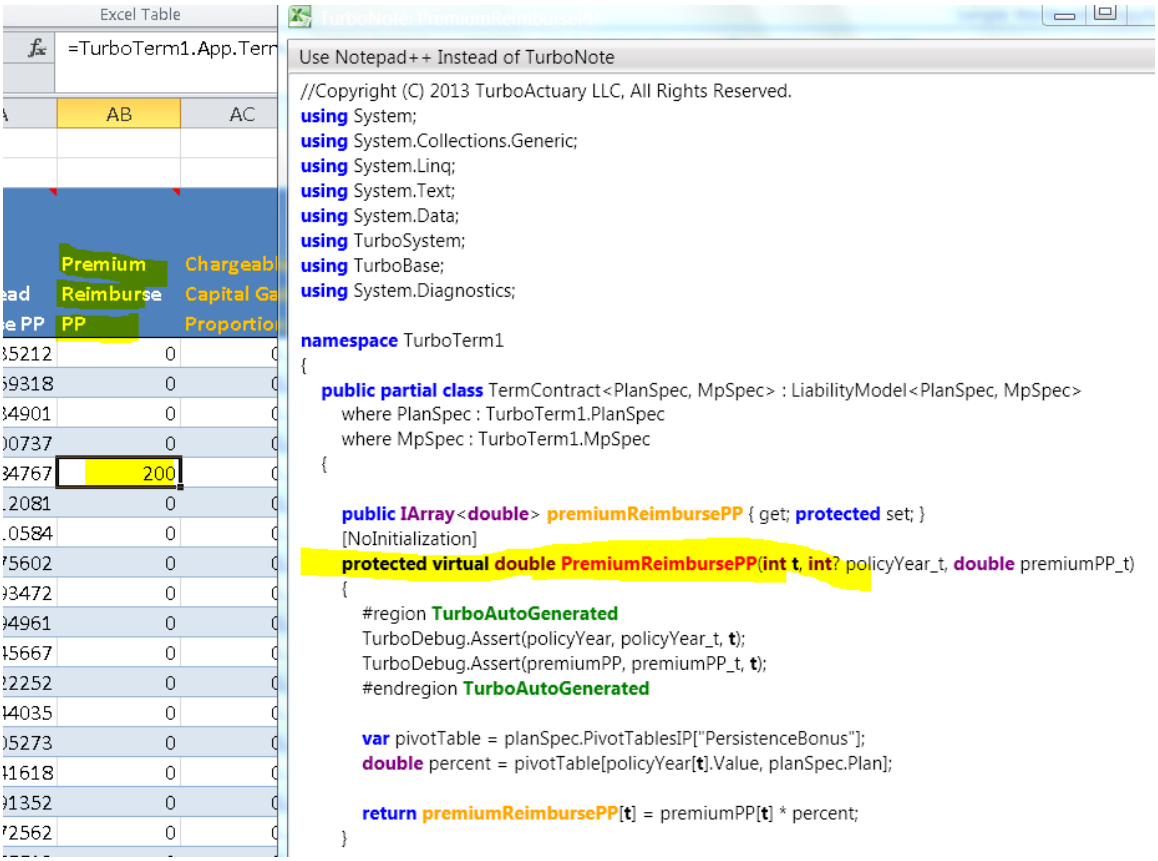
a) To view and debug your code in Visual studio:

Please follow chapter 4 of Quick Start Guide – TurboExcel.

b) To view code in TurboNote:

Please first follow section 2.2 of this document to generate .SourceCode file for the enhanced model. So please click the the submenu 'Generate .SourceCode for View C# Code" from **Visual Studio**.

Then follow the section 2.3 of Quick Start Guide – TurboExcel.



5.9 Output Results

In the Visual Studio model (platform model), you can output your results to csv file or database (Sql Server/Oracle etc).

In the Main function of Program.cs, you can specify what you want to output:

```
DataTable outputTable = DataOutput.CreateTable(
    new ColumnSpec("Product", typeof(string)),
    new ColumnSpec("Plan Group", typeof(string)),
    new ColumnSpec("Plan", typeof(string)),
    "Book Profit", "Embedded Value");
```

Default type is 'double'

```
foreach (DataRow planRow in planToRun.AsEnumerable())
{
    ProductEnum product = Utility.ToEnum<ProductEnum>(planRow.Field<string>("Product"));
    PlanGroupEnum planGroup = Utility.ToEnum<PlanGroupEnum>(planRow.Field<string>("Plan Group"));
    PlanEnum plan = Utility.ToEnum<PlanEnum>(planRow.Field<string>("Plan"));

    PlanSpec planSpec = new PlanSpec(dataAccessLayer, "Term1", globalSpec, product, planGroup, plan, thisRun);

    DataTable modelPointRows = ModelPointUtility.GetModelPoints(dataAccessLayer, "ModelPointTable", plan);


    IArray<double>[] embeddedValues = new IArray<double>[modelPointRows.Rows.Count];
    IArray<double>[] bookProfits = new IArray<double>[modelPointRows.Rows.Count];

    for (int i = 0; i < modelPointRows.Rows.Count; i++)
    {
        MpSpec mpSpec = new MpSpec(globalSpec, planSpec, modelPointRows.Rows[i], product, planGroup, plan);

        Term term = new Term(globalSpec, planSpec, mpSpec, 0, globalSpec.TotalProjectionMonths,
            mainScenarioManager, scenarioNumber, true);
        term.Calculate();

        embeddedValues[i] = term.embeddedValue;
        bookProfits[i] = term.bookProfit;
    }

    DataOutput.InsertIntoTable(outputTable, 0, int.MaxValue,
        new ColumnContent("Product", product.FastToString()),
        new ColumnContent("Plan Group", planGroup.FastToString()),
        new ColumnContent("Plan", plan.FastToString()),
        new ColumnContent("Book Profit", bookProfits.Sum()),
        new ColumnContent("Embedded Value", embeddedValues.Sum()));
}

DataOutput.ToCsvFile(@"c:\TurboActuary3.0\output\TurboTerm1.csv", outputTable);
```

Here we output only bookProfits and embeddedValue. You can choose to output more columns.

Appendix 1: Coding Reuse/Inheritance

In C#, inheritance enables us to extend the functionality of one class to the other class that can be reused and modified accordingly. The class whose members are inherited by the other class is called the **base class** and the class that inherits the members of base class is called the **child class**.

The following example show that Dog class inherits/reuses the "Eating" function from parent class Animal, and we add a new function "Barking" into the child class Dog.

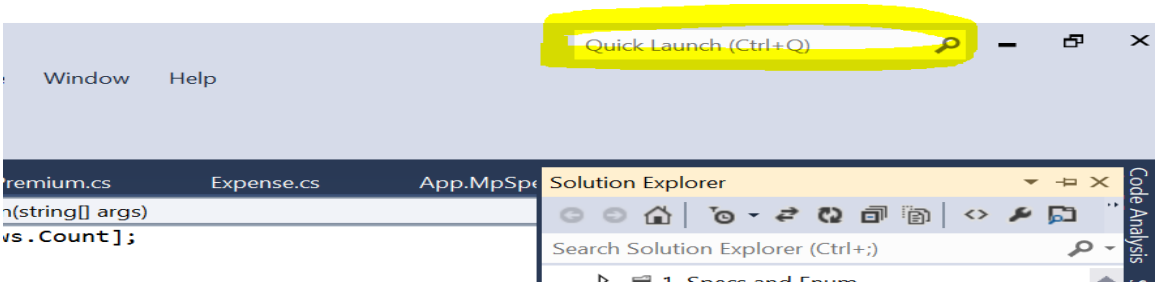
```
public class Animal
{
    public virtual void Eating()
    {
        Console.WriteLine("Every animal can eat!");
    }
}


public class Dog : Animal
{
    public virtual void Barking()
    {
        Console.WriteLine("Besides eating, dog can bark");
    }
}


class InheritanceExample
{
    static void Main(string[] args)
    {
        Animal animal = new Animal();
        animal.Eating();  //I can only eat

        Dog dog = new Dog();
        dog.Eating();  //I can eat
        dog.Barking(); //I can bark
    }
}
```

Appendix 2: How to Use Visual Studio

A good site to learn Visual Studio for beginner is: http://www.learnvisualstudio.net/start-here/ or consult your IT colleagues.

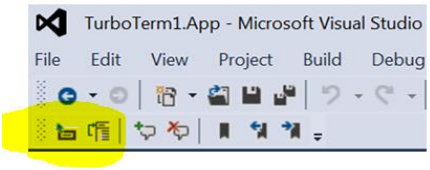
2.1 Quick Launch in VS 2012

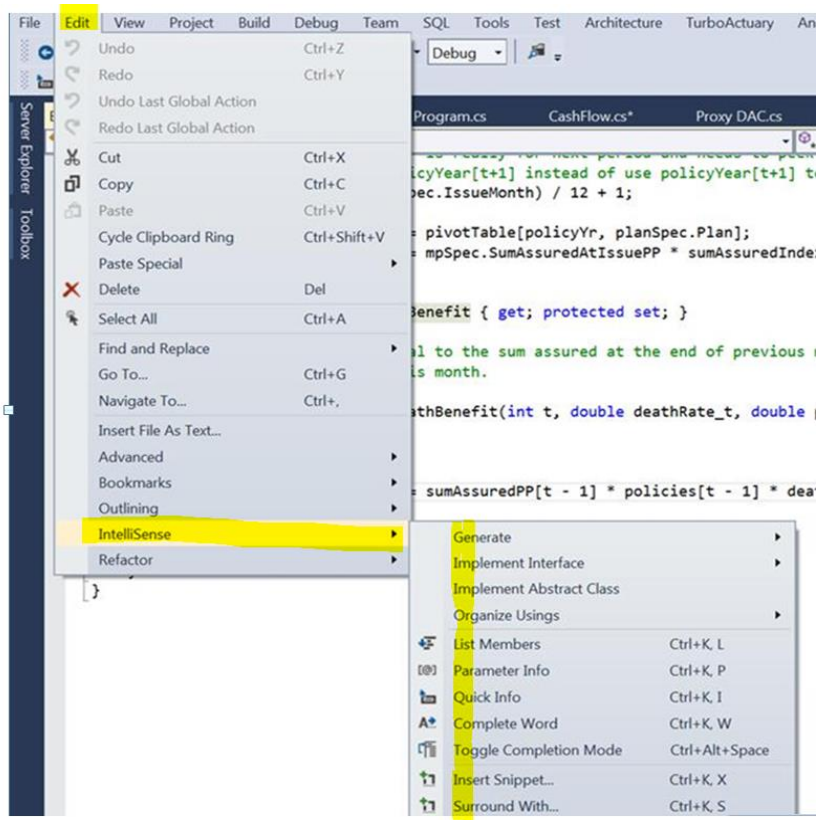
Even when you know where the option or menu item lives, it's often quicker to hit Ctrl+Q and type a few letters to find a command or option.



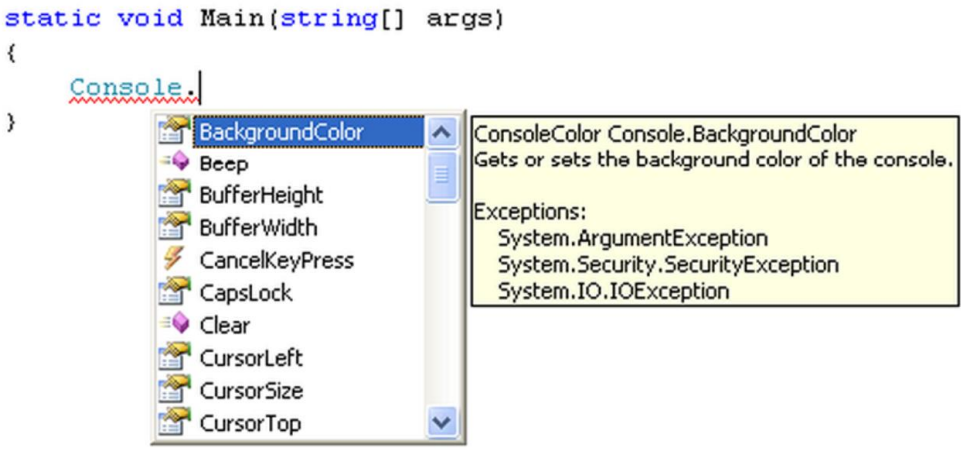
2.2 IntelliSense

IntelliSense is the general term for a number of features: List Members, Parameter Info, Quick Info, and Complete Word, etc. These features help you to learn more about the code you are using, keep track of the parameters you are typing, and add calls to properties and methods with only a few keystrokes.



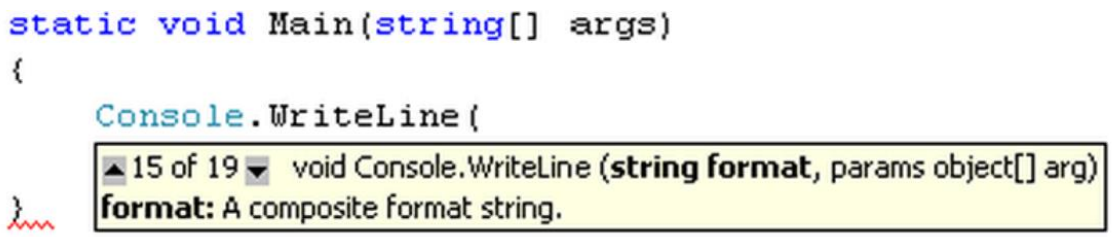


List Member: you can get a list of valid members of a type like class by typing its name followed by "." as shown below:



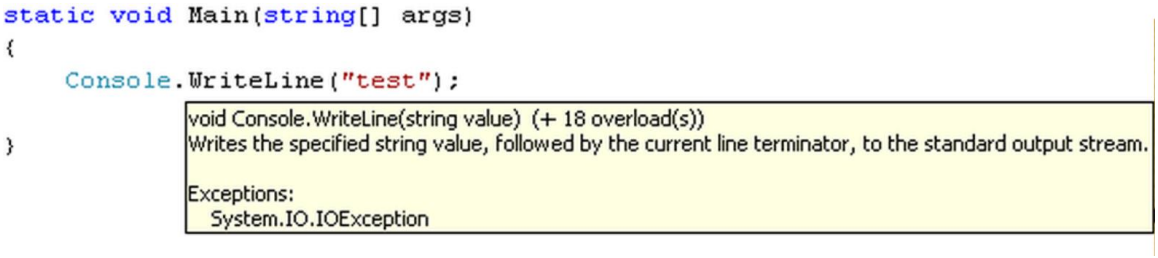
By default, Auto list members will be enabled. We can manually List members of a type by pressing Ctrl+J.

Parameter Info: you can get information about parameters like its type, name of parameters for a method as shown below:



You can invoke Parameter Info manually by using Ctrl+Shift+Space.

Quick Info: you can display the complete declaration of an identifier as shown below:



You can get Quick Info of an identifier by placing a cursor over it. You can manually invoke it by Ctrl+K, Ctrl+I.

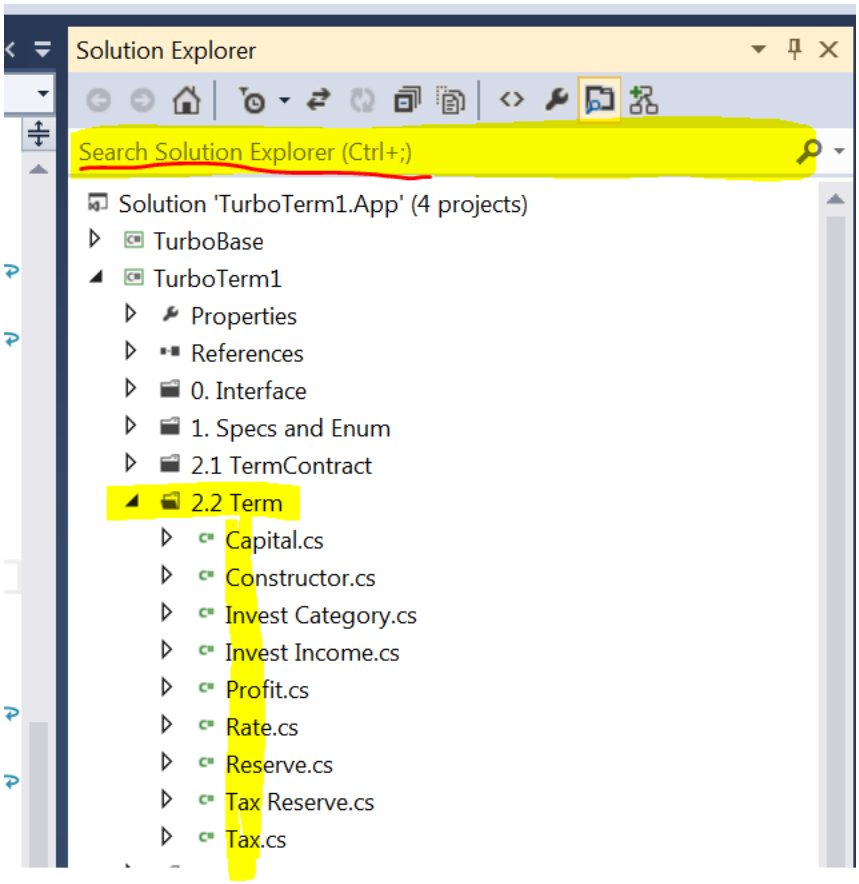
Complete Word: it will help you to complete a word without typing full name of it. You can type first few letters of an identifier and press Ctrl+SpaceBar to complete the word. If we have multiple matches, it will show list of matching words in List Members popup.

2.3 Coding Navigation

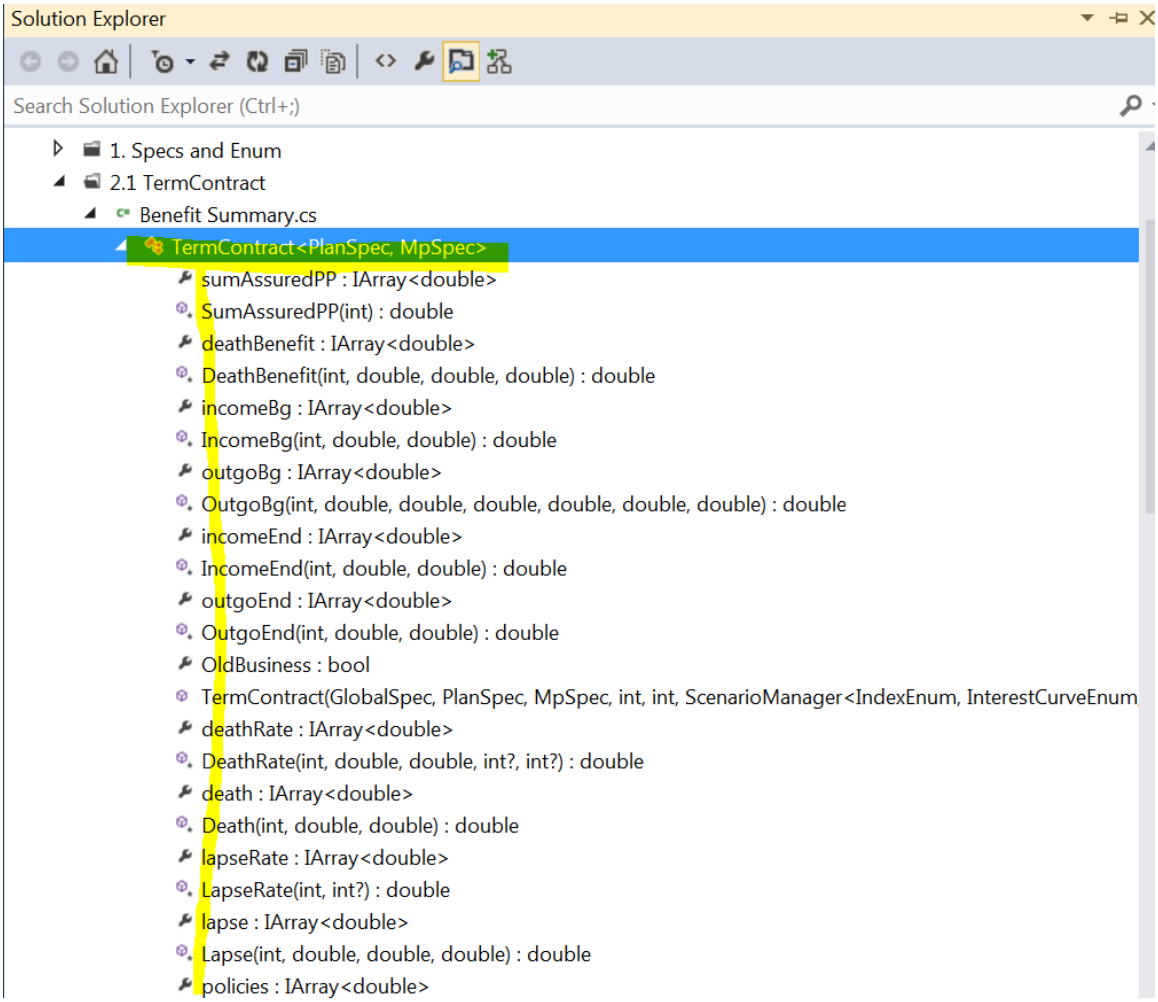
We use VS 2012 as examples in this section. VS 2010 should be similar with some differences.

As usual, you can use “Ctrl+f”, “Ctrl+Shift+f” to search. In addition, Visual Studio provides great flexibility to tailor your needs.

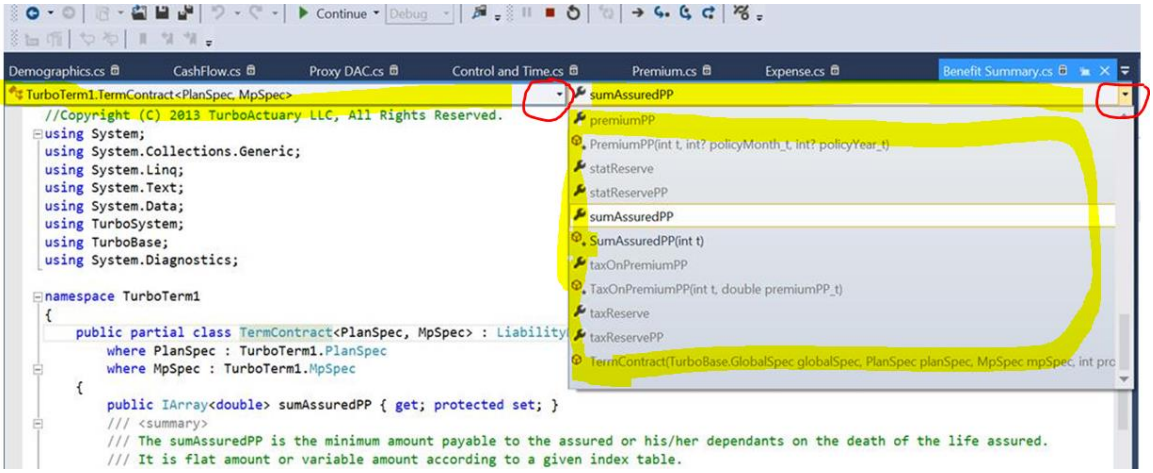
You can use **Solution Explore** to find your files under a class quickly. The file names are used as category names in TurboX.



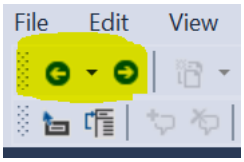
You can also use Solution Explorer to find all methods (functions), properties (data with get/set) and fields (data) under a class quickly.



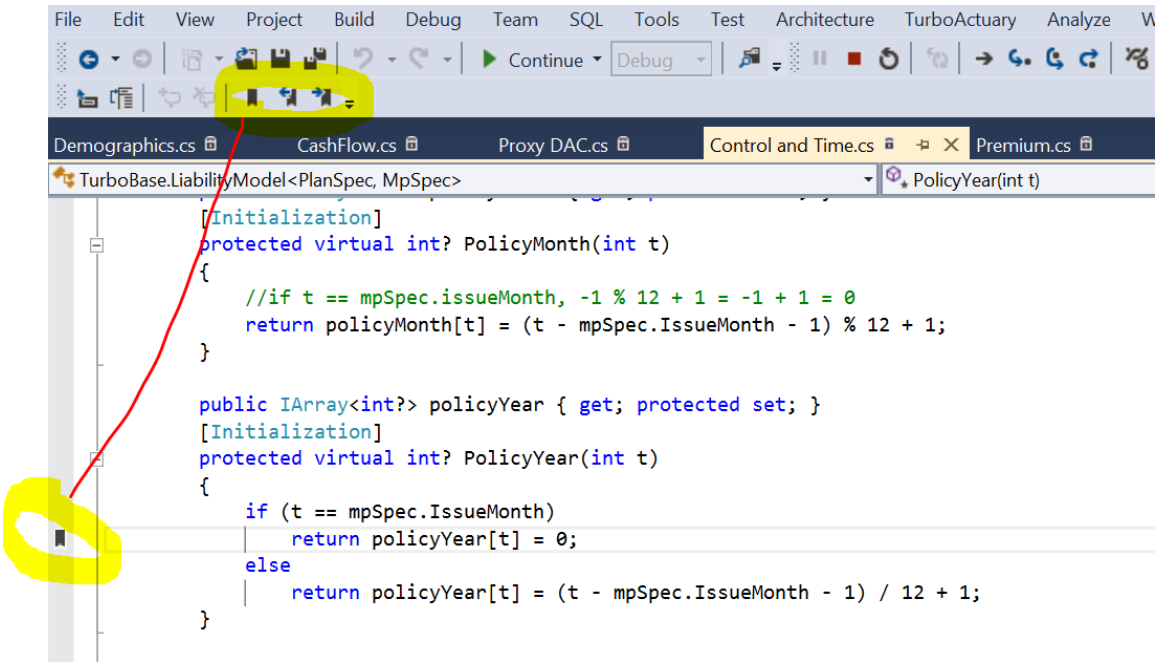
Or you can use **Explorer Bar** to find all methods (functions), properties (data with get/set) and fields (data) under a class quickly.



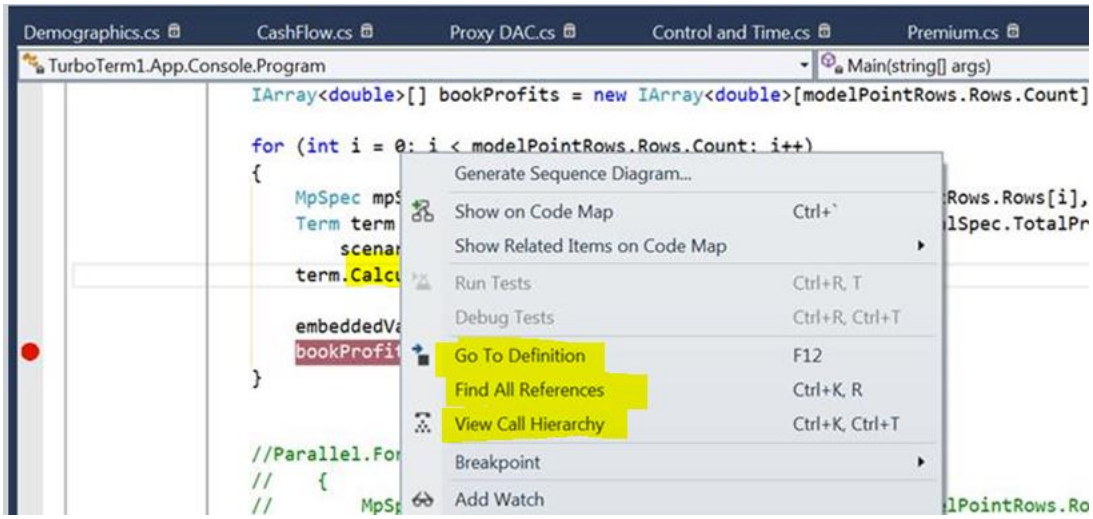
You can use **back and forth buttons** to navigate codes just as you did in Internet Browser (Explorer, Chrome, or Firefox):

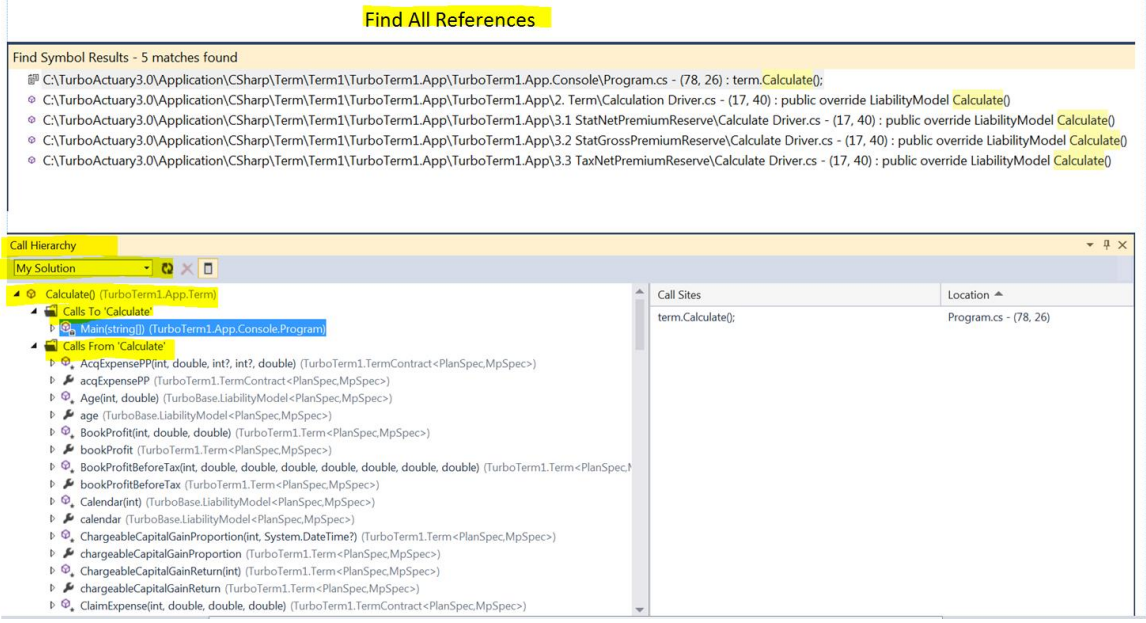


Or do **bookmark** the same way as your Internet Browser so you can visit it late using bookmarks:

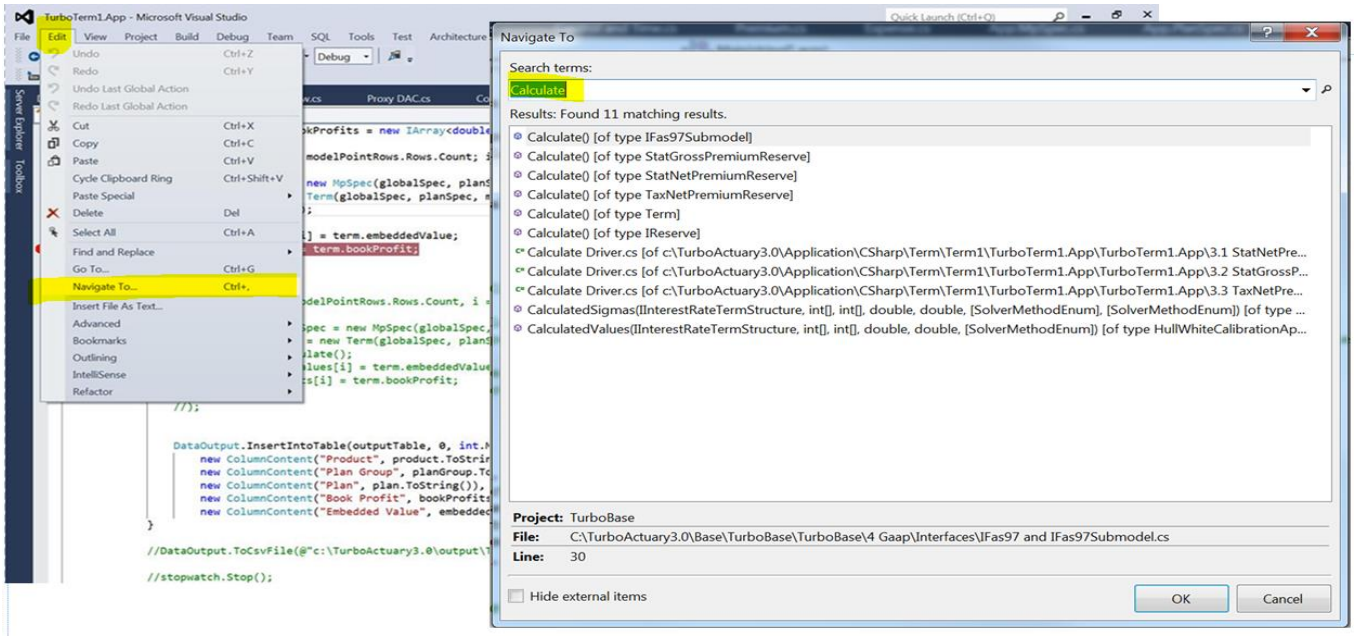


You can **go to definition**, **find all references**, or **view call hierarchy** from the context menu by right-clicking on a code element (method, property, field, or constructor name) in the code editor.



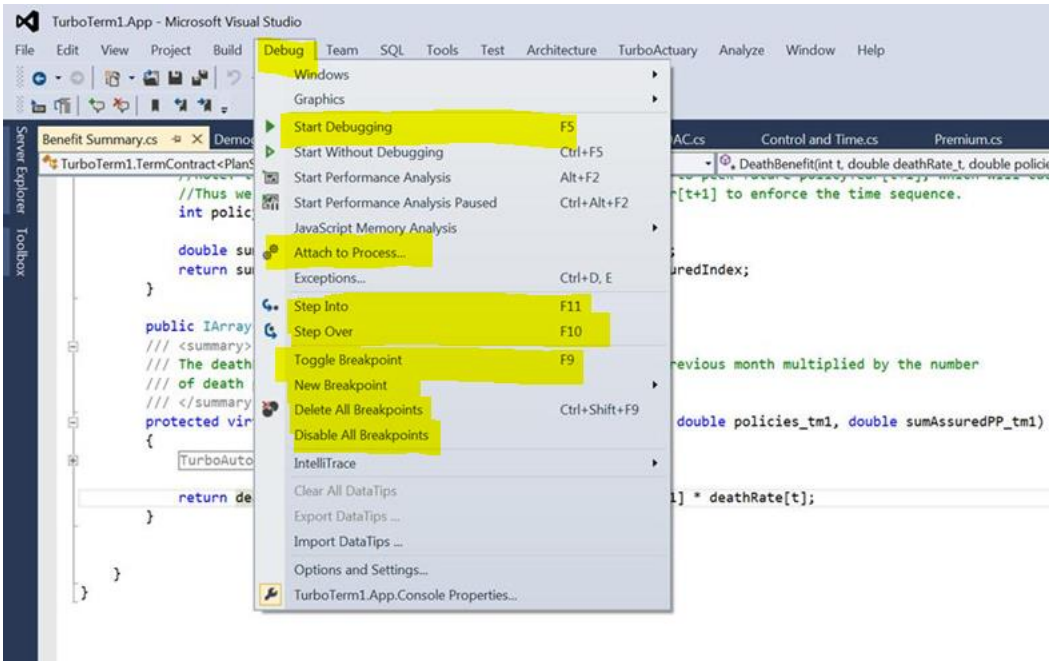

Find All References
Find Symbol Results - 5 matches found
C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App\TurboTerm1.App.Console\Program.cs - (78, 26) : term.Calculate();
C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App\TurboTerm1.App\2. Term\Calculation Driver.cs - (17, 40) : public override LiabilityModel Calculate()
C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App\TurboTerm1.App\3.1 StatNetPremiumReserve\Calculate Driver.cs - (17, 40) : public override LiabilityModel Calculate()
C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App\TurboTerm1.App\3.2 StatGrossPremiumReserve\Calculate Driver.cs - (17, 40) : public override LiabilityModel Calculate()
C:\TurboActuary3.0\Application\CSharp\Term\Term1\TurboTerm1.App\TurboTerm1.App\3.3 TaxNetPremiumReserve\Calculate Driver.cs - (17, 40) : public override LiabilityModel Calculate()
Call Hierarchy
My Solution
Calculate() (TurboTerm1.App.Term)
Calls To 'Calculate'
Main(string[]) (TurboTerm1.App.Console.Program)
Calls From 'Calculate'
Call Sites
Location
term.Calculate();
Program.cs - (78, 26)

You can use **Navigator To** (Ctrl+,) to search symbol definitions within a solution:



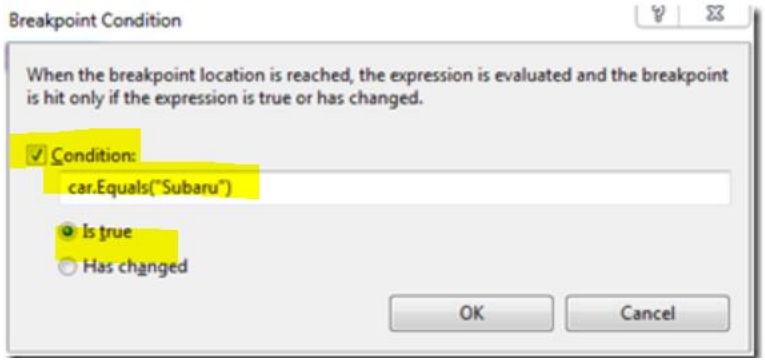
2.4 Debugging

A basic, but very good introduction can be found in http://www.dotnetperls.com/debugging.

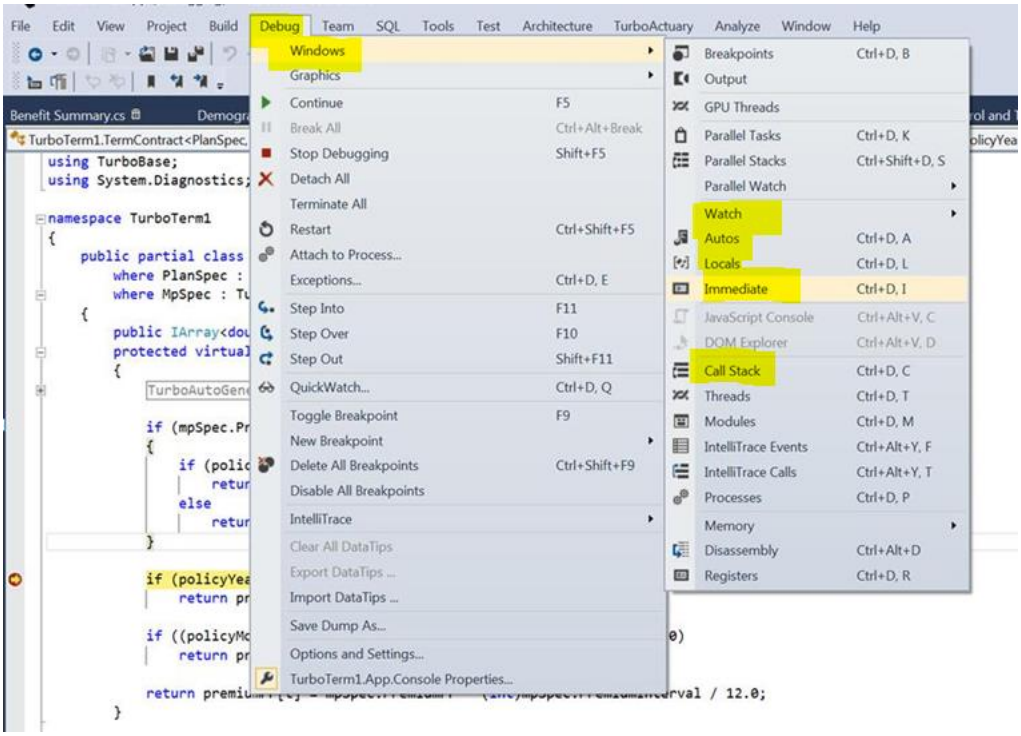


Conditional breakpoint allows you to set specific conditions that have to be met for a breakpoint to freeze execution. You have the choice between several types of conditions, like the value of a variable or the number of iterations over a specific breakpoint required before execution stops.

Select "Condition" which will open up the following dialog box:


Breakpoint Condition
When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.
Condition:
car.Equals("Subaru")
Is true
Has changed
OK
Cancel

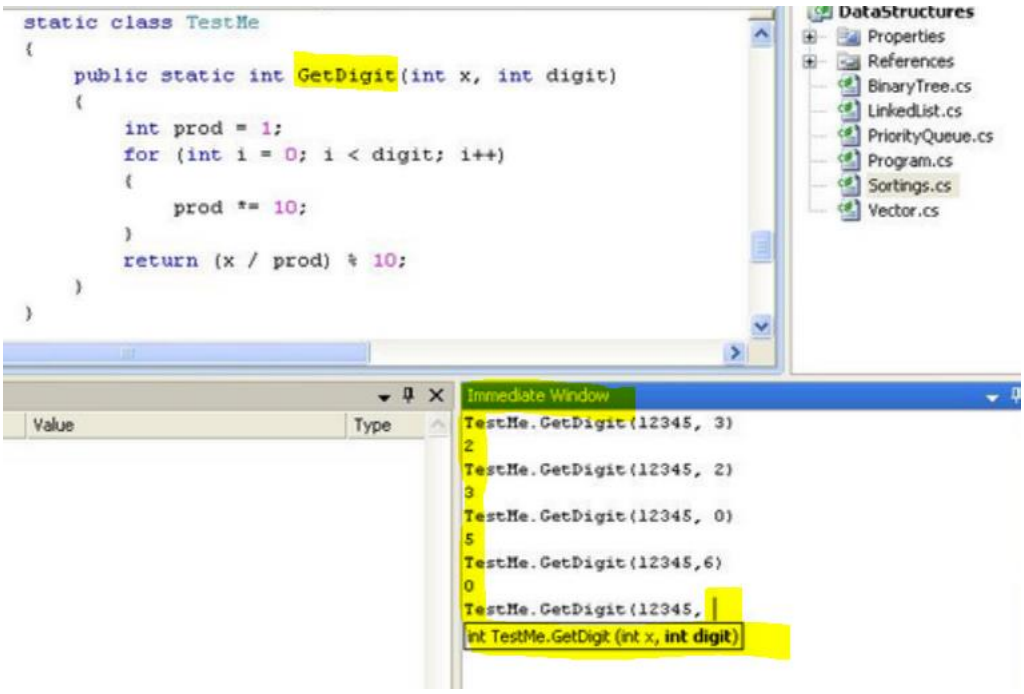
Debug windows are tool windows that help you debug.



Local Window: all local variables will be listed here, allowing you to get a quick overview of their name, type and value. You can even right-click in the grid and select "Edit value", to give a variable a new value. This allows you to test your code under other conditions than the current ones.

Watch Window: you can use the window to explore the object values, and call methods from watch window. You can refer to http://www.codeproject.com/Articles/121623/10-Tips-you-should-know-about-Watch-Window-While-d.

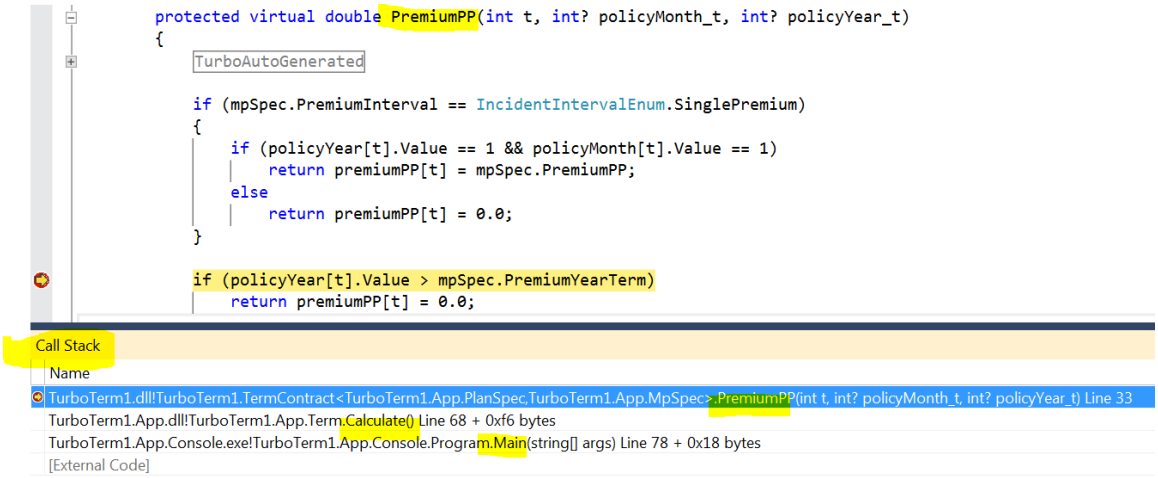
Immediate Window (Ctrl+Alt+i): it is a great tool to increase your productivity.



You can use the Immediate Window at design time also. The cool thing is that now you're able to try out some code without firing up the debugger – what a huge time saver!

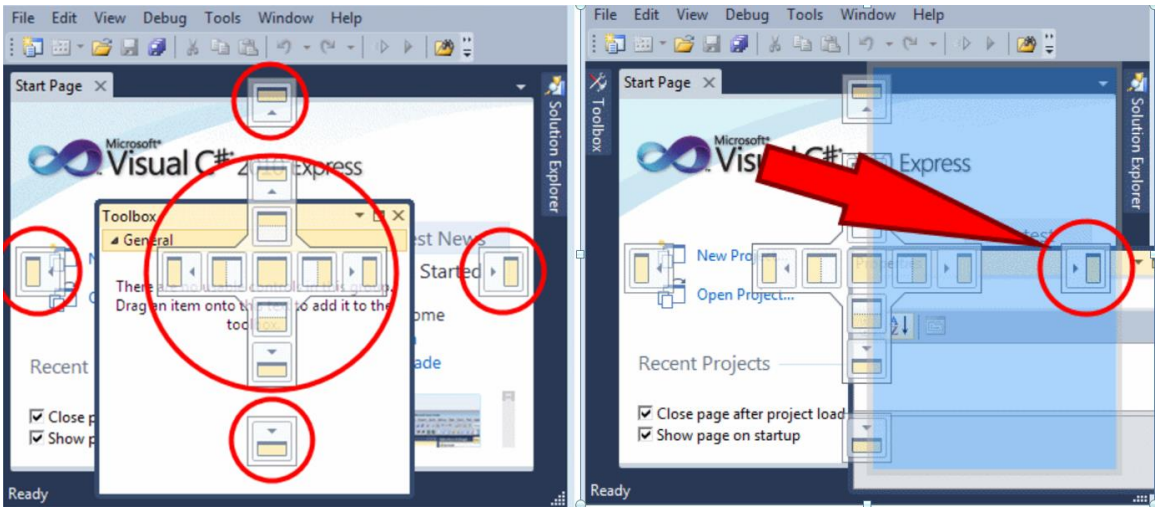
```
Immediate Window
int j = 2;
2
? j
2
double v = System.Math.Pow(9.0, j);
81.0
? v
81.0
```

Call Stack (Ctrl+Alt+c): from the call stack window, you can find who is calling the function, and cascade down to the main function. It is a great debugging tool.



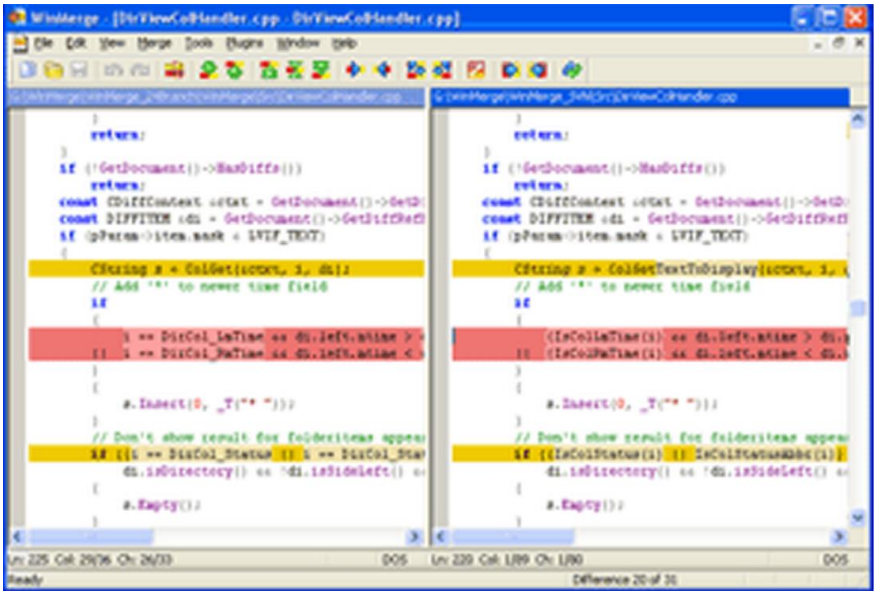
2.5 Docking Window

Visual Studio allows developers to move, resize, and pin these windows (edit window, toolbox windows, solution explorer windows, etc.) inside Visual Studio's top-level window, or to float them and move them anywhere on the desktop, **including to a secondary monitor**. Please refer to http://www.functionx.com/csharp/Lesson01.htm.



2.6 Diff Tool

You can integrate external programs to perform your file merging and file diffing activities.



2.7 Useful Shortcut

Ctrl + E,D	Auto format the document
Ctrl+,	Open the "Navigate to" dialog
Ctrl+-	Navigate backwards in the cursor position history
Ctrl+Shift+-	Navigate forwards in the cursor position history
Ctrl+M,O	Collapse to definitions
Ctrl+Shift+Space	To bring up the intellisense suggestions
F12	Go to Definition
Shift+F12	Find all references
Ctrl + x	Delete current selection or current line

Ctrl + Space	Completes the current word in the completion list
Ctrl + Alt + Space	Suggestion mode
Ctrl + j	Open IntelliSense completion list
Alt + Mouse Up/Down	Multiline Editing
Ctrl+Shift+Up/Down	Highlight References
Ctrl+Mouse Scrool	Zoom
Ctrl + F3	Search for the current selection

You can check http://stackoverflow.com/questions/98606/favorite-visual-studio-keyboard-shortcuts and see more shortcuts people are using.

Appendix 3: About C#

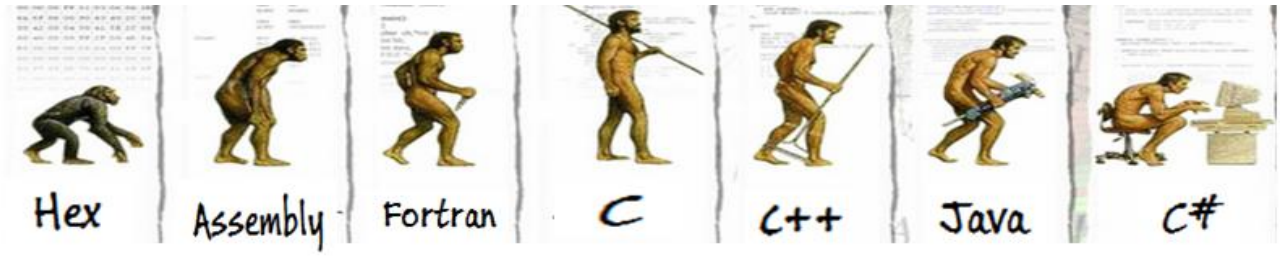
C# is the most popular language in windows platform, and was developed and is fully supported by Microsoft. According to http://en.wikibooks.org/wiki/C_Sharp_Programming, it has features that allow beginners to become proficient in C# more quickly than in C or C++. Similar to Java, it is object-oriented.

You only need to know basic C#. You can study by examples or read some C# tutorial from Internet.

A good book about C# is "C# in a Nutshell", but you only need to read chapter 2, chapter 3, and section nullable types in chapter 4.

A good reference is http://msdn.microsoft.com/en-us/library/aa645596(v=vs.71).aspx.

You can visit http://stackoverflow.com and get helps from C# community.



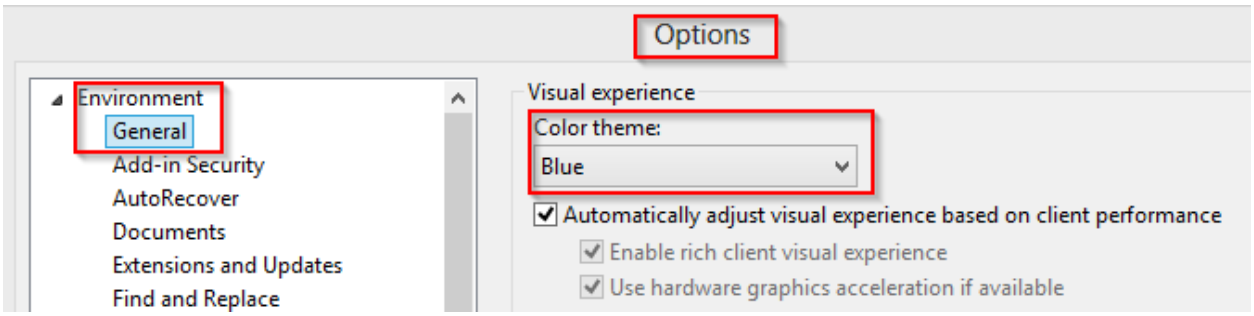
Appendix 4: Visual Studio Tools

4.1 Blue Theme (Optional)

You can customize VS 2012 menu colors (many people like the blue theme).

First, you need to update your VS 2012. Go to Tools menu, then Extensions and Updates, then Updates, then Product Updates, and click Update.

After update, start VS2012, and go to Tools menu, then Environment, then General, and choose blue in the color theme.



4.2 Indent Guides (Optional)

It adds vertical lines at each indent level in Visual Studio 2010/2012 as:

```
public virtual double GmabMoneyness(int t, double avPP_t, double gmabPP_t, double gmabPresentValuePP_t)
{
    TurboAutoGenerated

    if (!mpSpec.GmabIndicator)
        return gmabMoneyness[t] = 0.0;

    switch (planSpec.GmabMoneynessDefinition)
    {
        case MoneynessDefinition.GuaranteeLumpSum:
            if (avPP[t].EpisionLarger(0.0))
                return gmabMoneyness[t] = gmabPP[t] / avPP[t];
            else
                return gmabMoneyness[t] = 1000.0; //or any large number

        case MoneynessDefinition.GuaranteePresentValue:
            if (avPP[t].EpisionLarger(0.0))
                return gmabMoneyness[t] = gmabPresentValuePP[t] / avPP[t];
            else
                return gmabMoneyness[t] = 1000.0; //or any large number

        default:
            throw new Exception("error");
    }
}
```

It is free and can be downloaded from http://visualstudiogallery.msdn.microsoft.com/e792686d-542b-474a-8c55-630980e72c30.

After installation, start VS 2012, and go to Tools menu, then Options submenu, find "Indent Guides", and set as the following picture.